On a Neutral Network-Based Fault Detection Algorithm

by

Mufeed Ahmed Saleh Al-Ghumgham

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

SYSTEM ENGINEERING

July, 1992

Order Number 1354117

On a neural network-based fault detection algorithm

Al-Ghumgham, Mufeed Ahmed Saleh, M.S.

King Fahd University of Petroleum and Minerals (Saudi Arabia), 1992

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

ON A NEURAL NETWORK-BASED FAULT DETECTION ALGORITHM

Mufeed Ahmed Saleh AL-Ghumgham

SYSTEMS ENGINEERING

JULY 1992

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

This thesis, written by

Mufeed Ahmed Saleh AL-Ghumgham

under the direction of his thesis committee, and approved by all the members, has been presented to and accepted by the Dean, College of Graduate Studies, in partial fulfillment of the requirement for the degree of

MASTER OF SCIENCE IN SYSTEMS ENGINEERING

Thesis Committee

Chairman (Dr. J. Ezzine)

Member (Dr. M. Ben Daya)

Member (Dr. H.E. Shabaik)

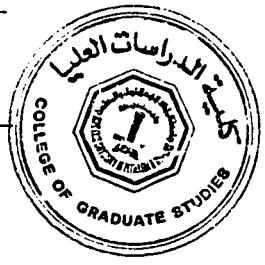
Member (Dr. M. Bettayeb)

Member (Dr. R. Braham)

Department Chairman

Dean, College of Graduate Studies

Date : 5-7-97



Dedicated to

My Parents

and

Family

Acknowledgments

All praise be to Allah, the Lord of the worlds. May peace and blessings be upon Mohammad the last of the messengers and his family. I thank Allah for his limitless help and guidance.

I gladly acknowledge the generous help and support for this research given by the King Fahd University of Petroleum & Minerals.

I wish to extend my appreciation and gratitude to my advisor Dr. J. Ezzine, and to other committee members Drs. M. Ben Daya, H. Shabaik, M. Bettayeb, and R. Braham. I would like also to thank Dr. A.U. Al-Kaabi, from Research Institute.

My thanks to the graduate students of the Department of Systems Engineering and my friends S. Al-Amer, M. Dikko, A. Baig, I. Akin, U. Sekerday, Y. Al-Sulais, and A. Al-Marzooq from whom I learned a lot and who made the long work hours pleasant.

Contents

List of Tables

List of Figures

خلاصة الرسالة

اسم الطالـــب : مفيد احمد صالح الغمغام

عنوان الرسالة : طريقة للتعرف على المشاكل والعيوب بواسطة الشبكات العصبيه الاصطناعية

التخصـــص : هندسة نظم وأساليب

تاريخ الشهادة : يوليو ١٩٩٢ م

في السنوات الأخيره ، ظهرت عدة أساليب وطرق للتعرف على المشاكل والعيوب في الأنظمة الديناميكيه . ومن هذا الأساليب المستخدمه أسلوبان هامان هما مراقب ليونبرجر الثابت والمراقب الحساس ، ووظيفتهما هي عبارة عن أعطاء أنذار بحصول خلل ما في أية نظام ومن ثم تصحيح هذا الخلل بأسرع ما يمكن . في هذا البحث سنتعرض ألى كيفية حصول أنذار خاطيء ، عند استخدام مراقب ليونبرجر الثابت والمراقب الحساس. هناك اسباب لحدوث الانذار الخاطيء منها وجود اخطاء في النموذج الرياضي او وجود مشوشات تؤثر على النظام .

في هذة الاطروحه تُستخدم الشبكات العصبيه الاصطناعية جنبًا الى جنب مع الاسلوبين السابقين للحصول على طريقه جيده لاكتشاف الاخطاء والعيوب في الانظمه الديناميكيه مع تقليل الانذارات الخاطئه . وقد تم اختبار هذه الطريقه لاكتشاف التسربات من اربع خزانات متصله .

درجة الماجستير في العلوم

جامعة الملك فهد للبترول والمعادن

الظهران ، المملكة العربية السعودية

يوليو ١٩٩٢ م

Abstract

Name: Mufeed Ahmed Saleh AL-Ghumgham
Title: On a Neural Network-Based Fault Detection Algorithm
Major Field: Systems Engineering
Date of Degree: July 1992

In recent years, many techniques have been proposed for the detection and isolation of abrupt changes in dynamical systems. Two of these schemes, namely, robust observers and detection filter, are studied in this thesis. The sensitivity of these approaches to parameter variations, or modeling errors or both, makes them unable to detect faults and causes false alarms in the detection logic.

In this thesis, a potential solution is presented. It involves the use of neural networks along with the current observer-based scheme. With this approach, one can achieve a robust failure detection scheme with minimal sensitivity to parameter perturbation, system's non-linearities, and white noise. A four-tank non-linear system is used for illustration.

Master of Science Degree
King Fahd University of Petroleum & Minerals
Dhahran, Saudi Arabia
July 1992

Chapter 1

INTRODUCTION

1.1 Introduction

Nowadays, dynamical systems such as process control plants, are becoming larger and more complex. Thus, any abrupt failure, such as sensor failure, actuator failure, or component failure, will cause a shutdown or a down time to these systems. Hence, early failure detection, can help avoid major plant breakdowns and catastrophies, that could otherwise result in substantial material damage and even human fatalities. Thus, the purpose of failure detection and diagnosis is to allow an early detection and isolation of any abrupt failure.

In some applications, such as edge detection, the effect of an abrupt failure is direct and simple - a bias develops in output signal. In other applications, such as chemical processes, the effect of an abrupt failure could be very critical, and is usually described in a more complex and indirect manner, e.g. a sudden change in the dynamics of the system. Such failures have severe consequences on systems,

for example, a leak in the fuel tank of the American space shuttle CHALLENGER resulted in the loss of the mission in 1986; a failure in the cooling system of an exothermic chemical reactor, results in an increase in the temperature of the reactor etc.

1.2 Fault Detection Schemes

There are several techniques used in fault detection and isolation. These approaches can be divided into two categories: qualitative and quantitative schemes. The qualitative approach uses techniques such as fault trees, signed directed graphs, fuzzy logic, and neural networks (Himmelblau [1], Hoskins et al. [2]).

The quantitative approach can be further classified into two types: stochastic and deterministic schemes. The stochastic scheme is basically based on the use of a single Kalman filter (Mehra et al. [3]) or banks of Kalman filters (Frank [4]). Some of these techniques are innovation scheme [3], generalized likelihood ratio (GLR) test, and multiple filters scheme (Basseville [5]). The deterministic scheme however, involves the use of Lunenberger observer. Two techniques of this approach will be studied in this thesis, namely, robust observers (RBO) and detection filter (DF). A wide variety of methods have been reviewed by Willsky [6], Isermann [7], and Frank [4].

1.3 Robust Observers and Detection Filter Schemes

Recently, many estimation techniques have received a great deal of attention. Two of these techniques are the robust observers approach (Ge and Fang [8]), and the detection filter (also known as sensitivity filter) approach (Beard [9], Jones [10], Viswanadham [11]).

The robust observers approach consists of two main steps: (i) generation of state estimates using a bank of reduced order observers, and (ii) a detection logic based on the observer errors.

The detection filters approach also consists of two main steps too: (i) generation of state estimates using a set of full-order observers, and (ii) a detection logic that is based on the directional properties of the observer errors. Unfortunately, the sensitivity of these two techniques to parameter variations and modeling errors, make them unable to correctly detect and isolate faults, resulting in frequent false alarms!

1.4 Objective of the Thesis

Parameter perturbations and modeling errors are unavoidable!. Consequently, it is mandatory that any solution technique used to control or monitor a given system be robust, and continue to perform satisfactorily within acceptable parameter perturbations and modeling errors.

One way to reduce the rate of false alarms while using robust observers or detection filters, is to use the method suggested by Frank and Keller [12]. Basically, this method consists of designing two full-order observers. The gain of the first observer is selected such that it is only sensitive to failures and insensitive to parameter perturbations, while the gain of the second is selected so that it is only sensitive to parameter perturbations and insensitive to failures. The major difficulty of this approach is the lack of a systematic design algorithm to select these two gains.

The objective of this thesis is to robustify the standard robust observers, and detection filter techniques via artificial neural networks, i.e. reduce false alarms due to parameter variations, or modeling errors or both. Indeed, neural networks have been applied to fault detection and isolation in dynamical systems. The feasibility of this approach has been shown in several applications, such as a fluidized catalytic cracking unit (Venkatasubramanian et al. [13]), and chemical reactor [10].

Nevertheless, as shown in this thesis, artificial neural networks can be inappropriate for certain systems, especially those with slow dynamics such as the four-tank non-linear system presented in this thesis. In fact, for this class of systems, the robust observers, or detection filter techniques can be used quite successfully in absence of parameter perturbations. Consequently, to take advantage of the strengths of standard techniques along with the "intelligence" of artificial neural networks, a hybrid solution design technique is proposed in this thesis. As

a matter of fact, the decision logic used in the standard techniques is the major source of sensitivity. The back-propagation artificial neural networks are used in this novel scheme to implement this decision logic.

1.5 Outline of the Thesis

The thesis is organized as follows. Chapter two will outline the design procedures for two failure detection and isolation schemes: robust observers and detection filter. In Chapter three, a four-tank non-linear system is used to illustrate the design procedures for both techniques. The sensitivity of both the robust observers and the detection filter to parameter perturbations is investigated in Chapter four, and the results are applied to the four-tank non-linear system. In Chapter five, the observer-based neural networks approach is presented and then applied to the four-tank non-linear model. Summary, conclusions and recommendations for future work are presented in Chapter six.

Furthermore, two appendices will be used in this thesis. In Appendix A, the four-tank non-linear system is derived. Appendix B, presents a simulation package used to illustrate the different failure detection approaches used throughout this thesis (this package is run with MATLAB software).

Chapter 2

FAULT DETECTION AND ISOLATION VIA OBSERVER-BASED TECHNIQUES

2.1 Introduction

The purpose of this chapter is to present two well known design procedures for failure detection and isolation, namely, robust observers (RBO) and detection filter (DF) algorithms. The RBO technique involves the design of a bank of reduced-order Lunenberger observers, to detect and monitor any abnormality in the dynamic system (Ge and Fang [8]). Some of these observers are designed, such that, they become sensitive to some of these faults while robust to the remaining faults. Any failure that occurs, is detected and isolated through a binary decision logic.

The DF scheme, however, consists of a full-order Lunenberger observer, with a special choice of the observer gain D. It is selected such that the residual vector (i.e. output error between the DF and the system) has certain directional properties at the occurrence of a certain fault. This was first proposed by Beard [9], and Jones [10].

This chapter is organized as follows: section two presents the robust observers. The detection filter will be discussed in section three. The following points will be presented in both sections: (i) structure of the observers (or observer), (ii) detection logic, (iii) design procedures, and (iv) design considerations. Conclusions will be drawn in section four.

2.2 The Robust Observers (RBO)

The robust observers are a bank of reduced order observers, that are used to monitor the operation of the system. In this section, the full design procedure will be outlined. The design algorithm is from Ge and Fang [8].

2.2.1 Structure of RBO

In general, a linear time-invariant system with possible k failure modes can be described as follows:

$$\dot{x}(t) = Ax(t) + Bu(t) + \sum_{i=1}^{k} L_i m_i(t), \tag{2.1}$$

$$y(t) = Cx(t), \quad (\dot{}) = \frac{d}{dt} \tag{2.2}$$

where $x(t)$ is the n-dimensional state, $u(t)$ is the input, and $y(t)$ is the m-dimensional output. L_i's represent the k failure modes. L_i's are known matrices called failure signature matrices, and m_i's are unknown signals called failure functions. When failure i occurs the elements of L_i are linearly independent.

The general structure of the detection observer (F, G, T, j, p) has been derived by Ge and Fang [8] as follows

$$\dot{z}(t) = Fz(t) + Gy(t) + TBu(t), \tag{2.3}$$

$$e(t) = j^T z(t) + p^T y(t) \tag{2.4}$$

The parametric matrices F, G, T, j, and p are subject to the structure conditions

$$TA - FT = GC, \tag{2.5}$$

$$j^T T + p^T C = 0, \tag{2.6}$$

$$F \textit{ be stable} \tag{2.7}$$

where $e(t)$ of (2.4) is called an observation signal.

2.2.2 Detection Logic of RBO

To isolate component failures via robust observation, one has to construct a bank of detection observers each of which is sensitive to only $(k - m + 1)$ failure modes, while insensitive to $(m - 1)$ of these failure modes. The total number of robust

observers is given by $\begin{pmatrix} n \\ m-1 \end{pmatrix}$. The four-tank example will be considered to illustrate this algorithm.

2.2.3 Design Procedures of RBO

The design of fault detection observers, amounts to determining the parameters (F, G, T, j, p). From [8], the solution of the structure conditions (2.5), (2.6) and (2.7) are summarized in the following theorem.

Theorem I

Suppose rank (C) = m, let

$$F = \begin{bmatrix} s & & & \\ 1 & s & & 0 \\ & \ddots & \ddots & \\ & 0 & 1 & s \\ & & & 1 \quad s \end{bmatrix} \tag{2.8}$$

then the solution for the sturcture conditions is given as follows

$$T_i = \sum_{j=0}^{i-1} \frac{1}{j!} V_{i-j} C \frac{d^j}{ds^j} R(s), \quad i = 1, 2, ..., d, \tag{2.9}$$

$$G = -p(F)V, \tag{2.10}$$

$$j^T T Ker(C) = 0, \tag{2.11}$$

$$p = -KT\,C^T (CC^T)^{-1}, \tag{2.12}$$

where $T \in \Re^{d\times n}$, $F \in \Re^{d\times d}$, $V \in \Re^{d\times m}$, d is the observer order of the detection observer, T_i is the ith row of T, V_j is the jth row of V, $p \in \Re^{n\times 1}$, $j \in \Re^{d\times 1}$, and

$$R(s) = det(sI_n - A)(sI_n - A)^{-1} = \sum_{i=1}^{n} a_i \sum_{j=0}^{i-1} s^j A^{i-j-1}, \tag{2.13}$$

$$p(s) = det(sI_n - A) = a_n s^n + a_{n-1}s^{n-1} + ... + a_1 s + a_0 \tag{2.14}$$

$$V = [\ V_1 \quad V_2 \quad ... \quad V_d\ ]^T, \tag{2.15}$$

$$T = [\ T_1 \quad T_2 \quad ... \quad T_d\ ]^T \tag{2.16}$$

Based on the results of theorem 1, an efficient design algorithm from [8] is outlined below to design a detection observer sensitive to a set of (m-1) specified component failure modes.

Step I Calculate a_i's, $i = 0, 1, 2, ..., n$.

Step II Specify $(m-1)$ integer numbers i_1, i_2, ..., i_{m-1}, according to robustness requirements such as the ones shown in table 3.3.

Step III Choose an eigenvalue s for the detection observer, where s does not belong to the spectrum of A. Let the observer order be d, and start with $d = 1$.

Step IV Calculate $R(s)$ and $M(s)$ as follows:

$$R(s) = det(sI_n - A)(sI_n - A)^{-1},$$

$$M_o = CR(s)$$

M_o has n columns denoted by $(M_o)_i$, with $i = 1, 2, ..., n$.

Let S_o be formed as follows

$$S_o = [\ (M_o)_{i_1} \quad (M_o)_{i_2} \quad \ldots \quad (M_o)_{i_{m-1}}\ ]$$

Solve

$$V_1 S_o = 0$$

for $V_1 \in \Re^{1 \times m}$ and $V_1 \neq 0$, calculates

$$T_1 = V_1 M_o$$

Step V Find $Ker(C) = 0$, (i.e. the null space of C) as follows:

$$CL^T = 0$$

Solve for L^T.

Step VI Compute(TL^T); where T is formed according to (2.16).

Step VII If rank $(TL^T) < d$, then go to step IX, else go to step VIII.

Step VIII Increase the observer order by one:

$$d = d + 1$$

Calculate M_{d-1} as follows

$$M_{d-1} = \frac{1}{d!} C R^{(d-1)}(s)$$

Let

$$S_{d-1} = \left[\ (M_{d-1})_{i_1} \quad (M_{d-1})_{i_2} \quad \ldots \quad (M_{d-1})_{i_{m-1}}\ \right]$$

Solve

$$V_d S_o = -\sum_{k=1}^{d-1} V_{d-k} S_k$$

for V_d, then calculate

$$T_d = \sum_{k=0}^{d-1} V_{d-k} M_k$$

Go to step VI.

Step IX Solve for j, using (2.11). Let F be chosen as in (2.8). Calculate p from (2.12). Find G from the following equation (assuming that $m \leq n$).

$$G = (TA - FT)\, C^T (CC^T)^{-1}$$

Step X End the algorithm if all detection observers with robustness required have been met, else go to step II to design a detection observer robust to another set of component failure modes.

2.2.4 Design Considerations of RBO

In order to determine the number of robust observers needed to monitor the dynamical system, one first has to determine all failure modes. Next, determine the model of the process. For a non-linear process, linearization of the system around an operating point is needed, and then the design procedure is carried out. Standard design considerations have to be adhered to.

2.3 The Detection Filter (DF)

The Fault detection filter (DF) is a full-order observer, that is used to detect abrupt faults, such as component faults, and actuator faults. It was first proposed by Beard [9] and later Jones [10] extended some of the results in [9] and gave a thorough procedure for modeling failures and designing a DF. In this section, the main design steps will be outlined. The number of DF used to detect the k failure modes, is in general less then the number of RBO. Also, the observer gain D of the DF, is selected such that the residual error has certain directional properties at the occurrence of a certain fault.

2.3.1 Structure of DF

Consider the following dynamical system described by the state space model under nominal condition.

$$\dot{x}(t) = Ax(t) + Bu(t) + \sum_{i=1}^{k} L_i m_i(t), \tag{2.17}$$

$$y(t) = Cx(t) \tag{2.18}$$

where u, x, and y are p, n, and m dimensional respectively, and A, B, and C are constant matrices of appropriate sizes.

The DF is described by

$$\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t) + D(y(t) - C\hat{x}(t)), \tag{2.19}$$

$$\hat{y}(t) = C\hat{x}(t) \tag{2.20}$$

where $\hat{y}$, $\hat{x}$, and u are p, n, and m dimensional respectively, $D \in \Re^{n \times m}$, and k is

the total number of failure modes.

Now, defining $\dot{e}(t) = \dot{x}(t) - \dot{\hat{x}}(t)$, leads to

$$\dot{e}(t) = (A - DC)e(t) + \sum_{i=1}^{k} L_i m_i(t), \tag{2.21}$$

$$\overline{e}(t) = -JCe(t) \tag{2.22}$$

Hence, $\overline{e}(s)$ can be found using the Laplace transform as follows

$$\overline{e}(s) = -JC(sI_n - A + DC)^{-1} \sum_{i=1}^{k} L_i m_i(t) \tag{2.23}$$

In the detection filter approach, D is chosen such that the residual vector $e(t)$ has certain directional properties when failure modes occur.

2.3.2 Detection Logic of DF

In order to detect and isolate the various failure modes, the following logic has to be implemented. Once the ith failure occurs, then the residual error $e(t)$, is orthogonal to $(k-1)$ failure signature matrices, while it is parallel to the ith failure signature matrix (L_i).

Step I Check if $\mid \overline{e} \mid < \epsilon$, then let $\theta_i = 90$, for $i = 1, ..., n$, else go to steps II and III. ϵ is a threshold value on the residual error. It is selected such that it takes into account system modeling errors, noise, and small parameter variations.

Step II If $\mid \bar{e} \mid \geq \epsilon$, then compute the angles between $\bar{e}$ vector and the failure signatures (L_i) as follows:

$$\theta_i = Cos^{-1}\left(\frac{\bar{e}^T L_i}{\mid \bar{e} \mid\mid L_i \mid}\right), \quad for\ i = 1, 2, ..., n. \tag{2.24}$$

where

$$\bar{e} = \left[\begin{array}{cccc} \bar{e}_1 & \bar{e}_2 & ... & \bar{e}_n \end{array}\right]^T,$$
$$\mid \bar{e} \mid = \sqrt{\bar{e}^T.\bar{e}} = \left\{\sum_{i=1}^{n}(\bar{e}_i)^2\right\}^{1/2}$$

Then, go to step III.

Step III Then, the angles θ_i's are interpreted as follows:

$$\theta_i = \begin{cases} \mid \theta_i \mid \leq \theta_t, & failure\ i\ occurred \\ 90, & normal\ operation \end{cases} \quad for\ i = 1, 2, ..., n \tag{2.25}$$

where

ϵ = *Threshold value on the error,*

θ_t = *Threshold angle value*

2.3.3 Design Procedure of DF

In order to design a DF, the following procedure is followed, which is valid for $m \leq n$.

Step I Find indices K_1, K_2, ..., K_k such that

$$CA^j L_i = 0, \quad j = 0, 1, ..., K_i - 1 \tag{2.26}$$

and

$$A^{k_i} L_i = 0, \tag{2.27}$$

where L_i is the ith failure mode signature vector. When $CL_i = 0$, then the ith failure produces no effect on the output $y(t)$.

Now define the $m \times k$ matrix L and the $n \times k$ matrix H given by

$$L = [\ CA^{K_1} L_1 \quad CA^{K_2} L_2 \quad \ldots \quad CA^{K_k} L_k\ ], \tag{2.28}$$

$$H = [\ A^{K_1+1} L_1 \quad A^{K_2+1} L_2 \quad \ldots \quad A^{K_k+1} L_k\ ] \tag{2.29}$$

It is assumed that L has full column rank k, i.e. matrix L has a left L^+ such that $L^+ L = I$, given by

$$L^+ = (L^T L)^{-1} L^T \tag{2.30}$$

Let L_n be the basis for the nullspace of L, i.e.

$$L_n L = 0 \tag{2.31}$$

It is noted that $L^+ \in \Re^{k \times m}$, and $L_n \in \Re^{(k-m) \times k}$ respectively. Let J be the nonsingular $k \times k$ matrix defined by

$$J = \begin{bmatrix} L^+ \\ L_n \end{bmatrix} \tag{2.32}$$

and let the filter gain matrix $\overline{D}$ be given by

$$\overline{D} = H L^+ \tag{2.33}$$

Step II Calculate the following matrices

$$\overline{A} = A - \overline{D}C, \tag{2.34}$$

$$\overline{F} = [\ L_1 \quad L_2 \quad \ldots \quad L_k\ ], \tag{2.35}$$

$$\overline{C} = JC \tag{2.36}$$

Step III Form the Q_i matrix as follows

$$Q_i = [\ \overline{f}_i \quad \overline{A}\,\overline{f}_i \quad \ldots \quad \overline{A}^{K_i}\overline{f}_i\ ] \tag{2.37}$$

where $\overline{f}_i$ is the ith column of $\overline{F}$.

Define

$$T = \left[\ Q_1 \quad Q_2 \quad \ldots \quad Q_k \quad Q_{k+1} \quad Q_{k+2}\ \right], \tag{2.38}$$

$$\tilde{A} = T^{-1}\overline{A}T, \tag{2.39}$$

$$\tilde{F} = T^{-1}\overline{F}, \tag{2.40}$$

$$\tilde{C} = \overline{C}T \tag{2.41}$$

where Q_{k+1} and Q_{k+2} are matrices chosen to make T have a rank equals to n, and $\tilde{A}$, $\tilde{F}$, and $\tilde{C}$ are matrices that have the following simple structure:

$$\tilde{A} = \left[\begin{array}{ccc|cc} \tilde{A}_1 & & & A_1^c & A_1^u \\ & \tilde{A}_2 & 0 & A_2^c & A_2^u \\ & 0 & \ddots & \vdots & \vdots \\ & & \tilde{A}_k & A_k^c & A_k^u \\ \hline 0 & \ldots & 0 & \tilde{A}_{k+1} & A_{k+1}^u \\ 0 & \ldots & 0 & 0 & \tilde{A}_{k+2} \end{array}\right],$$

$$\tilde{F} = diag\left(\tilde{f}_1, \tilde{f}_2, ..., \tilde{f}_k\right),$$

$$\tilde{C} = \left[\begin{array}{ccc|cc} \tilde{C}_1 & & & C_1^c & C_1^u \\ & \tilde{C}_2 & 0 & C_2^c & C_2^u \\ & & & & \\ & 0 & \ddots & \vdots & \vdots \\ & & \tilde{C}_k & C_k^c & C_k^u \\ \hline 0 & \cdots & 0 & 0 & \tilde{C}_{k+2} \end{array}\right],$$

where $\tilde{A}_i$, $\tilde{f}_i$, and $\tilde{C}_i$ are $q_i \times q_i$, $q_i \times 1$ and $1 \times q_i$ vector matrices respectively. q_i is the rank of the matrix Q_i. $\tilde{A}_{k+1}$, $\tilde{A}_{k+2}$ and $\tilde{C}_{k+1}$ are $q_{k+1} \times q_{k+1}$, $q_{k+2} \times q_{k+2}$ and $(m-k) \times q_{k+2}$ dimensional matrices.

Step IV Choose the detection filter gain matrix $\tilde{D}$, to stabilize the canonically decoupled system. Choose $\tilde{D}$ to be

$$\tilde{D} = \left[\begin{array}{ccc|c} \tilde{d}_1 & & & D_1^c \\ & \tilde{d}_2 & 0 & D_2^c \\ & & & \\ & 0 & \ddots & \vdots \\ & & \tilde{d}_k & D_k^c \\ \hline 0 & \cdots & 0 & D_{k+1} \\ 0 & \cdots & 0 & D_{k+2} \end{array}\right],$$

The choices of $\tilde{d}_1$, ..., $\tilde{d}_k$ and $\tilde{D}_{k+2}$ are important, while others are arbitrarily chosen. Poles have to be chosen such that $(\tilde{A}_i - \tilde{d}_i\tilde{C}_i)$ and $(\tilde{A}_{k+2} - \tilde{D}_{k+2}\tilde{C}_{k+2})$ have desired eigenvalues.

Define

$$t_i(s) = \mid sI - \tilde{A}_i + \tilde{d}_i\tilde{C}_i \mid;\ i = 1, 2, ..., k.,$$
$$t_{k+2}(s) = \mid sI - \tilde{A}_{k+2} + \tilde{D}_{k+2}\tilde{C}_{k+2} \mid$$

Step V Finally, compute the filter gain matrix D as follows

$$D = -HL^{+} + T\tilde{D}J \tag{2.42}$$

Now substituting D in (2.23), will result in a decoupled system, that can be written as follows

$$e(s) = G(s)m(s), \tag{2.43}$$
$$m(s) = \begin{bmatrix} m_1(s) & m_2(s) & ... & m_k(s) \end{bmatrix}^T, \tag{2.44}$$
$$G(s) = \left[\begin{array}{ccc} G_1(s) & & \\ & G_2(s) & 0 \\ & 0 & \ddots \\ & & G_k(s) \\ \hline 0 & ... & 0 \end{array}\right], \tag{2.45}$$
$$G_i(s) = \phi_i g_i(s),\ for\ i = 1, 2, ..., k \tag{2.46}$$

where ϕ_i is a constant number, and $g_i(s)$ is a scalar rational function.

2.3.4 Design Considerations of DF

Given the dynamic plant along with the instrumentation details, the failure modes to be monitored have to be selected. Depending on the number of failure modes present, the number of directions and planer signatures are computed. Then, the

number of detection filters needed to monitor these faults id determined. In order to decrease the fault detection time, the poles of the filter have to be placed on the left side in the complex plane.

Chapter 3

FAULT DETECTION AND ISOLATION APPLIED TO A FOUR-TANK NON-LINEAR SYSTEM

3.1 The Four-Tank Linearized Model

In this chapter, two failure detection schemes, namely, the robust observers, and the detection filter, will be used to detect leaks in a four-tank non-linear system shown in figure A.1. This system is described by a non-linear model. Appendix A, summarizes the full derivation of this model, and the linearized model. The

linearized model in the state-space form is described as:

$$\delta\dot{x}(t) = \frac{1}{10}\begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \end{bmatrix} \delta x(t) + \begin{bmatrix} .2 \\ 0 \\ 0 \\ 0 \end{bmatrix} \delta u(t) \tag{3.1}$$

$$x(t) = x_n + \delta x(t) \tag{3.2}$$

where $x_n = [4, 3, 2, 1]^T$, δx, and δu are the deviations from the operating point. Taking $\delta u = 0$, and substituting for $\delta x(t) = x(t) - x_n$, leads to

$$\dot{x}(t) = \frac{1}{10}\begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \end{bmatrix} x(t) + \begin{bmatrix} 0.1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.3}$$

$$y(t) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x(t) \tag{3.4}$$

Now, equations (3.3) (3.4) represent the linearized dynamical system. Figure 3.1 shows the response of the non-linear model versus the linearized model.

3.2 Failure Modeling

In this system, there are two types of faults that may occur, namely, leakages and cloggings at the outlet of the tanks. In appendix A, it is shown how to model both

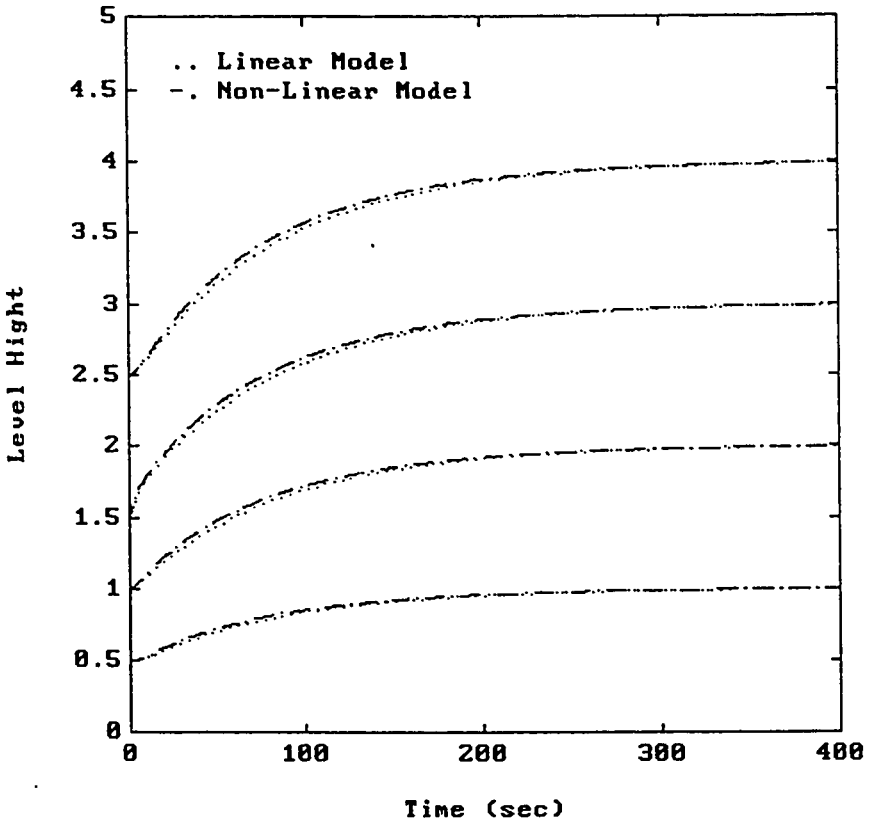


Figure 3.1: Response of the linear and non-linear models

types of faults. Thus the state equation with faults are represented as follows

$$\dot{x}(t) = F(x,u,t) + \sum_{i=1}^{4} L_i m_i(t) \tag{3.5}$$

where $F(x,u,t)$ is the non-linear model, L_i are the failure signatures of the leakages in tank i listed in table A.1, and $m_i(t)$ are the failure functions associated with the leakages in tank i listed in table A.2.

3.3 Robust Observers Design

In order to build a diagnosis system capable of detecting abrupt faults in the four-tank system, the linearized model will be used to design $\binom{n}{m-1}$ robust observers. Now since $n = 4$, then the number of observers are $\binom{4}{3} = 4$ and they are depicted in figure 3.2. Each detection observer is designed, such that, it is sensitive to only one failure mode, while insensitive to three failure modes.

3.3.1 Structure of RBO

Thus, the four robust observers are described as follows:

$$\dot{z}_i(t) = F_i z_i(t) + G_i y(t) + T_i B u(t), \tag{3.6}$$

$$e_i(t) = j_i^T z_i(t) + p_i^T y(t) \tag{3.7}$$

where e_i is the detection signal, and $Bu(t)$ is

$$Bu(t) = [\ .1 \quad 0 \quad 0 \quad 0\ ]^T \tag{3.8}$$

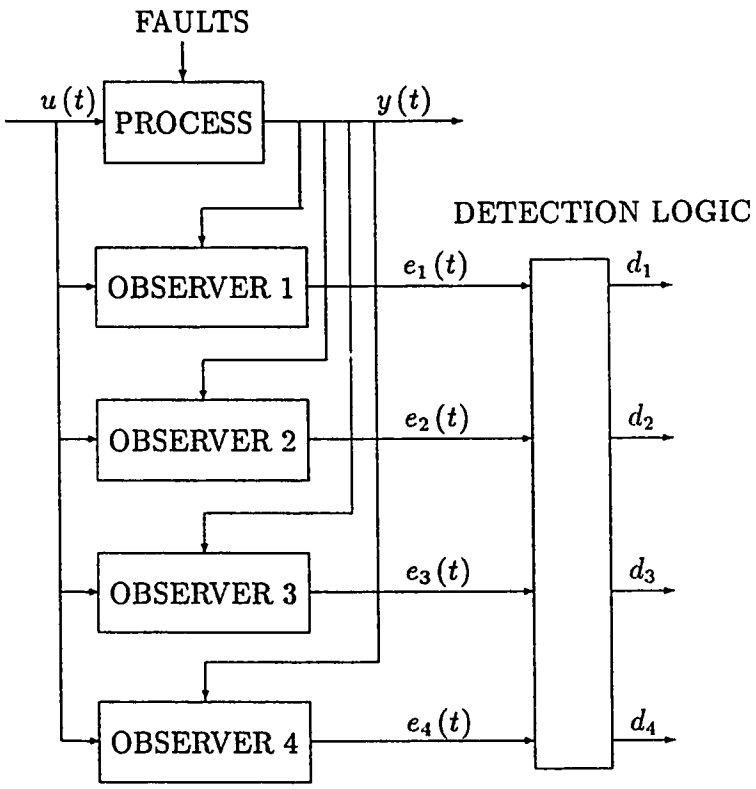


Figure 3.2: Fault detection via robust observers

3.3.2 Detection Logic of RBO

The detection logic that will be used to isolate these faults, will be constructed on the basis of table 3.3. Table 3.1 shows the sensitivity of each observation signal e_i, to the corresponding failure mode in state x_i.

Defining

$$f_i = \begin{cases} 1, & |e_i| \geq \lambda_i \\ 0, & |e_i| < \lambda_i \end{cases}, \quad for\ i = 1,2,3,4. \tag{3.9}$$

Where $\lambda_i > 0$ is a threshold to take account of unmodelled effects such as noise and non-linearities. Then, the four alarm signals can be defined by

$$d_i = f_i, \quad for \ i = 1,...,4 \tag{3.10}$$

where

$$d_i = \begin{cases} 1, & leakage\ in\ tank\ i \\ 0, & tank\ i\ OK \end{cases}, \quad for\ i = 1,2,3,4 \tag{3.11}$$

3.3.3 Design Procedure of RBO

In this part, the full design of the four observers will be carried out. The design procedure is based on the algorithm presented in section (2.2.3).

I. Find a_i's for $i = 0,1,...,4$, as follows

$$det(sI_4 - A) = s^4 + \frac{7}{10}s^3 + \frac{15}{10^2}s^2 + \frac{10}{10^3}s + \frac{1}{10^4} \tag{3.12}$$

Table 3.1: Robustness of observation signals to different failure modes

states	$e_1(t)$	$e_2(t)$	$e_3(t)$	$e_4(t)$
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

where I_4 is the identity matrix, and A is

$$A = \frac{1}{10}\begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \end{bmatrix} \tag{3.13}$$

Then,

$$a_o = \frac{1}{10^4},\ a_1 = \frac{1}{10^2},\ a_2 = \frac{15}{10^2},\ a_3 = \frac{7}{10},\ a_4 = 1 \tag{3.14}$$

II. Let the order of the observer be $d = 1$.

III. Let the observer poles be at $s = -2$.

IV. Calculate $R(s)$, and $M_o(s)$ as follows

$$R(s) = det(sI_4 - A)(sI_4 - A)^{-1} \tag{3.15}$$

$$R(s) = \begin{bmatrix} 5.7960 & 0.3230 & -0.0180 & 0.0010 \\ 0.3230 & -6.1370 & 0.3420 & -0.0190 \\ -0.0180 & 0.3420 & -6.1380 & 0.3410 \\ 0.0010 & -0.0190 & 0.3410 & -6.1190 \end{bmatrix} \tag{3.16}$$

Since $C = I_4$, then $M_o(s)$ is

$$M_o(s) = CR(s) = R(s) \tag{3.17}$$

V. The observers will now be designed to meet the robustness conditions listed in table 3.1.

e_1 observer:

1. Let $S_o = [\ (M_o)_2 \quad (M_o)_3 \quad (M_o)_4\ ]$ where $(M_o)_i$ is the ith column of M_o.

2. Next, calculate V_1 as follows

$$V_1 S_o = 0 \tag{3.18}$$

$$V_1 \begin{bmatrix} 0.3230 & -0.0180 & 0.0010 \\ -6.1370 & 0.3420 & -0.0190 \\ 0.3420 & -6.1380 & 0.3410 \\ -0.0190 & 0.3410 & -6.1190 \end{bmatrix} = [\ 0 \quad 0 \quad 0 \quad 0\ ] \tag{3.19}$$

where $V_1 \in \Re^{1\times 4}$.

Solving for V_1, leads to

$$V_1 = [\ -0.9986 \quad -0.0526 \quad 0 \quad 0\ ] \tag{3.20}$$

3. Calculate T_1, in the following way

$$T_1 = V_1 M_o = [\ 5.7710 \quad 0 \quad 0 \quad 0\ ] \tag{3.21}$$

4. Since $Ker(C) = 0$, then $L = [\ 0 \quad 0 \quad 0 \quad 0\ ]$

5. Since rank $(T_1 L) < d$, then $T = T_1$

6. Let $j_1 = 1$, and $F_1 = -2$, then

$$G_1 = (T_1 A - F_1 T_1) C^T (CC^T)^{-1} = [\ 10.9649 \quad 0.5771 \quad 0 \quad 0\ ] \tag{3.22}$$

$$p_1^T = -j_1^T T_1 C^T (CC^T)^{-1} = [\ -5.7710 \quad 0 \quad 0 \quad 0\ ] \tag{3.23}$$

For $e_i(t)$ robust observer, $i = 2, 3, 4$, one follows the same procedure. The final results are presented in table 3.2.

Table 3.2: $(F_i, j_i, T_i, G_i, p_i)$ Parameters for each RBO

$e_i(t)$	F_i	j_i	T_i^T	G_i^T	p_i
$e_1(t)$	-2	1	$\begin{bmatrix} 5.7710 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 10.9649 \\ 0.5771 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} -5.7710 \\ 0 \\ 0 \\ 0 \end{bmatrix}$
$e_2(t)$	-2	1	$\begin{bmatrix} 0 \\ -6.0813 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} -0.6081 \\ -10.9464 \\ -0.6081 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 6.0813 \\ 0 \\ 0 \end{bmatrix}$
$e_3(t)$	-2	1	$\begin{bmatrix} 0 \\ 0 \\ -6.0813 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ -0.6081 \\ -10.9464 \\ -0.6081 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 6.0907 \end{bmatrix}$
$e_4(t)$	-2	1	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ -6.0907 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ -0.6091 \\ -10.9632 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 6.0907 \end{bmatrix}$

3.3.4 Simulation Results

The performance of the failure detection scheme is tested and presented in figure 3.3. Four different faults are considered:

1. Leak in the base of tank 1 with a cross-section of 0.075 S_1.
2. Leak in the base of tank 2 with a cross-section of 0.1 S_2.
3. Leak in the base of tank 3 with a cross-section of 0.1 S_3.
4. Leak in the base of tank 4 with a cross-section of 0.1 S_4.

The system and the observers are simulated for a period of $100sec$. The faults occur at 50% of the simulation time. The simulation study is started by a step of $u = 1$. The initial conditions for the observers are chosen such that the observation signals start from zero. Figure 3.3 show the response of the alarm signals to a leak in tank 1. As can be seen, the leak can be clearly detected and isolated shortly after it happened. The alarm signals have picked out the right fault mode without ambiguity. The fault detection time for each fault is tabulated in table 3.3.

3.4 Detection Filter (DF) Design

In this section, a fault detection filter will be designed to monitor the four tank model carried out. The system with detection filter is depicted in figure 3.4.

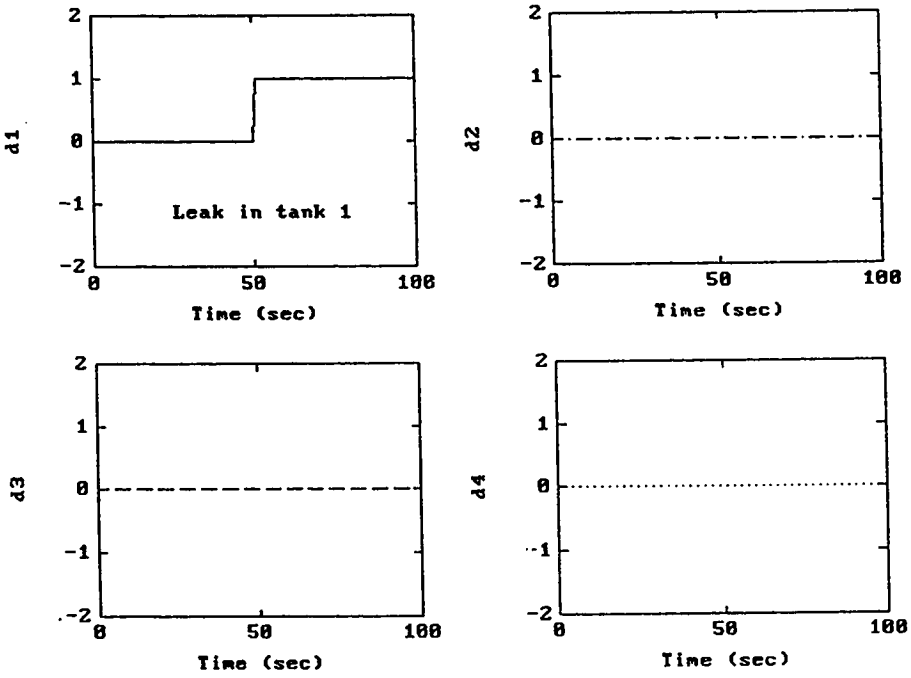


Figure 3.3: RBO alarm signals for a leak in tank 1

Table 3.3: RBO failure detection time for the four leaks

leak in tank i	detection time (sec)
1	0.54
2	0.40
3	0.60
4	1.30

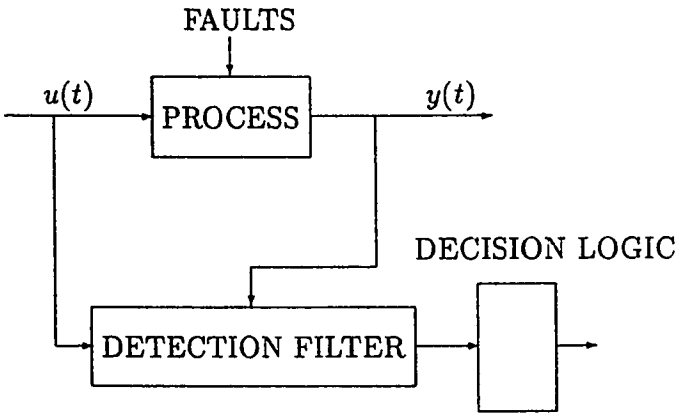


Figure 3.4: Fault detection via detection filter

3.4.1 Structure of the DF

The fault detection filter is described by

$$\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t) + D(y(t) - C\hat{x}(t)), \tag{3.24}$$

$$\hat{y}(t) = C\hat{x}(t) \tag{3.25}$$

where $C = I_4$, A and $Bu(t)$ are obtained from the linearized model (3.3).

The failure signatures are

$$L_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \; L_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \; L_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \; L_4 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{3.26}$$

3.4.2 Detection Logic of DF

In order to detect and isolate the various failure modes, the following logic has to be implemented.

Step I Check if $| \bar{e} | < \epsilon$, then let $\theta_i = 90$, for $i = 1, ..., 4$, else go to steps II and III.

Step II If $| \bar{e} | \geq \epsilon$, then compute the angles between the $\bar{e}$ vector and the failure signatures (L_i) as follows:

$$\theta_i = Cos^{-1}\left(\frac{\bar{e}_i}{| \bar{e} |}\right), \; i = 1, 2, 3, 4 \tag{3.27}$$

where

$$\bar{e} = [\bar{e}_1 \; \bar{e}_2 \; \bar{e}_3 \; \bar{e}_4]^T$$

$$| \bar{e} | = \sqrt{\bar{e}^T \bar{e}} = \left\{\sum_{i=1}^{4} (\bar{e}_i)^2\right\}^{1/2}$$

Then, go to step III.

Step III Then, the angles θ_i's are interpreted as follows:

$$\theta_i = \begin{cases} |\ \theta_i\ | \leq \theta_t & leakage\ in\ tank\ i \\ 90, & tank\ i\ OK \end{cases}, \quad for\ i = 1,2,3,4 \tag{3.28}$$

where

$$\theta_t = 10\ degrees$$

$$\epsilon = .05$$

3.4.3 Design Procedure of DF

Next, the design of the fault detection filter will be presented.

Step I To find the indices K_i for the four failure signatures, as follows:

$$K_1 = 0,\ K_2 = 0,\ K_3 = 0,\ and\ K_4 = 0 \tag{3.29}$$

Form the matrices L, H, L^+, L_n, and J leading to

$$L = [\ CL_1\ \ CL_2\ \ CL_3\ \ CL_4\ ] = I_4$$

$$H = [\ AL_1\ \ AL_2\ \ AL_3\ \ AL_4\ ] = A$$

$$L^+ = (L^T L)^{-1} L^T = I_4$$

$$L_n = Ker(C) = Ker(I_4) = 0$$

$$J = L^+$$

Then, calculate the gain matrix $\overline{D}$ as follows:

$$\overline{D} = HL^+ = A \tag{3.30}$$

Step II Calculate the following matrices

$$\begin{aligned} \overline{A} &= A - \overline{D}C = 0_4 \\ \overline{F} &= [\ L_1 \quad L_2 \quad L_3 \quad L_4\ ] = I_4 \\ \overline{C} &= JC = I_4 \end{aligned}$$

where 0_4 is a zero matrix of dimension (4×4).

Step III Forming the T matrix as follows:

$$T = [\ Q_1 \quad Q_2 \quad Q_3 \quad Q_4\ ] = [\ L_1 \quad L_2 \quad L_3 \quad L_4\ ] = I_4 \tag{3.31}$$

Then, $\tilde{A}$, $\tilde{F}$, and $\tilde{C}$ are

$$\begin{aligned} \tilde{A} &= T^{-1}\overline{A}T = \overline{A} \\ \tilde{F} &= T^{-1}\overline{F} = \overline{F} \\ \tilde{C} &= \overline{C}T = \overline{C} \end{aligned}$$

Step IV The gain matrix $\tilde{D}$, will be chosen such that $(\tilde{A} - \tilde{D}\tilde{C})$ have the desired eigenvalues. $\tilde{D}$ is a diagonal matrix given below:

$$\tilde{D} = diag(\tilde{d}_{11},\ \tilde{d}_{22},\ \tilde{d}_{33},\ \tilde{d}_{44}) \tag{3.32}$$

The characteristic polynomial of $(\tilde{A} - \tilde{D}\tilde{C})$ is as follows:

$$det(sI_4 - \tilde{A} + \tilde{D}\tilde{C}) = (s + \tilde{d}_{11})\ (s + \tilde{d}_{22})\ (s + \tilde{d}_{33})\ (s + \tilde{d}_{44}) \tag{3.33}$$

Let the desired eigenvalue be at $(-2.0, -2.0, -2.0, -2.0)$, then the elements of $\tilde{D}$ are chosen to be

$$\tilde{d}_{11} = 2.0,\ \tilde{d}_{22} = 2.0,\ \tilde{d}_{33} = 2.0,\ \tilde{d}_{44} = 2.0 \tag{3.34}$$

Step V Finally, the gain matrix D is computed as follows:

$$D = -HL^{+} + T\tilde{D}J = \frac{1}{10}\begin{bmatrix} 19 & 1 & 0 & 0 \\ 1 & 18 & 1 & 0 \\ 0 & 1 & 18 & 1 \\ 0 & 0 & 1 & 18 \end{bmatrix} \tag{3.35}$$

Thus, now define errors $e(t) = x(t) - \hat{x}(t)$ and $\overline{e}(t)$ to be

$$\dot{e}(t) = F(x,u,t) - A\hat{x}(t) - Bu(t) - D(Cx(t) - C\hat{x}(t)), \tag{3.36}$$

$$\overline{e}(t) = JC\,e(t) = e(t) \tag{3.37}$$

where

$$e(t) = [\ e_1(t) \quad e_2(t) \quad e_3(t) \quad e_4(t)\ ]^T$$

Now, the gain matrix D, will result in a decoupled system, i.e. the effect of one failure will propagate only in one direction.

3.4.4 Simulation Results

The performance of the detection filter was tested and is presented in figure 3.5. The faults considered here are the same faults simulated in section 3.3.3. As can be seen, the simulated faults occur at 50% of simulation time. In figure 3.5, the detection signals df_i's are plotted. These detection signals are based on the alarm

angles θ_i's. Hence, when one $df_i = 1$, then this means that $\theta_i \leq 10$ degrees. However, when $df_i = 0$, then this means that $\theta_i > 10$ degrees. In normal conditions, each element of θ_i approaches 90 degrees. However once a faults occurs, it is observed that only one element of θ_i crosses the threshold angle θ_t ($\theta_t = 10$ degrees) and reaches a steady state value. Hence, this allows a unique isolation of the faults, without ambiguity. The failure detection time is listed in table 3.4.

3.5 Conclusions

In this chapter, two diagnosis schemes have been used to detect and isolate leaks in a four-tank model. Four robust observers and one detection filter were designed. In a simulation study, both techniques have successfully detected a leak in tank 1.

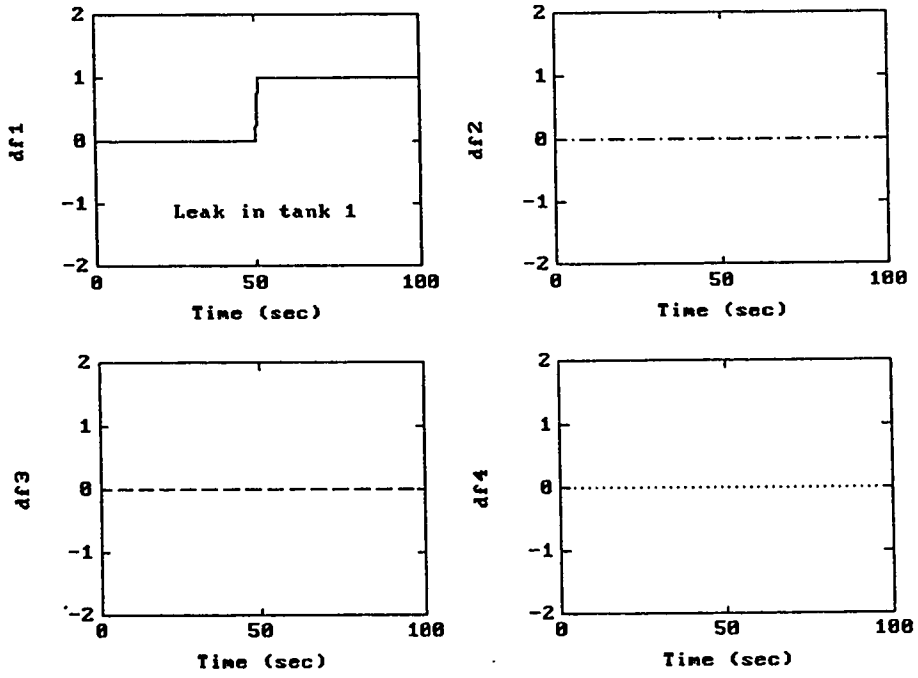


Figure 3.5: DF alarm signals for a leak in tank 1

Table 3.4: DF failure detection time for the four leaks

leak in tank i	detection time (sec)
1	0.51
2	0.50
3	0.50
4	0.51

Chapter 4

SENSITIVITY ANALYSIS OF FAULT DETECTION ALGORITHMS TO PARAMETER PERTURBATIONS

4.1 Introduction

The sensitivity of observer-based fault detection schemes to process parameter variations and/or modeling errors is one of the key concerns when it comes to their practical applicability. The parameters of the process, needed for observer design, are never known exactly, and they can also change with time. Conse-

quently, there is always a discrepancy between the actual parameters and those used in designing the observer. Therefore, once a failure alarm is declared, there is no way to find out whether it is a real fault or a false alarm due to parameter variations! False alarms are indicative of poor performance of the fault detection scheme, and this quickly leads to a lack of confidence in the detection algorithm. Thus, there is a pressing need to study the parameter sensitivity of observer-based fault detection schemes.

The purpose of this chapter, is to determine the range of acceptable parameter perturbations, due to parameter variations or modeling errors, beyond which false alarms are declared. This analysis is based on perturbation theory along the lines of Ezzine [18], Frank and Keller [12]. The problem is formulated in section two. Then solutions of the state equation, ith robust observers, and detection filter under parameter perturbations, are given in sections three, four, and five, respectively. A case study summary is presented in section six.

4.2 Problem Formulation

First, it is assumed that the process in question, is operating under nominal conditions. Standard linearization techniques, allow us to represent the dynamics of the process, about its nominal operating point, by linear time-invariant system.

$$\dot{x}_n(t) = A_n x_n(t) + B_n u(t), \tag{4.1}$$

$$y_n(t) = C_n x_n(t) \tag{4.2}$$

where $A_n \in \Re^{n\times n}$, $B_n \in \Re^{n\times p}$, $C_n \in \Re^{m\times n}$, $x_n \in \Re^{n\times 1}$, $y_n \in \Re^{m\times 1}$, and $u \in \Re^{p\times 1}$.

In the presence of process parameter perturbations the state equations of the process can be written as

$$\dot{x}(t) = (A_n + \Delta A)x(t) + (B_n + \Delta B)u(t), \tag{4.3}$$

$$y(t) = (C_n + \Delta C)x(t) \tag{4.4}$$

where $\Delta A \in \Re^{n\times n}$, $\Delta B \in \Re^{n\times p}$, and $\Delta C \in \Re^{m\times n}$.

Without loss of generality, it will be assumed henceforth, that the perturbation matrices are small enough to be represented in the following form

$$\Delta A = \epsilon\ \delta A, \tag{4.5}$$

$$\Delta B = \epsilon\ \delta B, \tag{4.6}$$

$$\Delta C = \epsilon\ \delta C \tag{4.7}$$

where ϵ is a small positive number. One can argue that the ϵ's, multiplying each of the above perturbation matrices, must be different! However, this can be partially accounted for via the perturbation matrices, provided that these perturbations are caused by the same parameter variations or modeling errors.

Consider now the ith robust observer used to detect the ith fault,

$$\dot{z}_i(t) = F_i z_i(t) + G_i y(t) + T_i B_n u(t), \tag{4.8}$$

$$e_i(t) = j_i^T z_i(t) + p_i^T y(t),\ \ i = 1, 2, ..., q \tag{4.9}$$

where $F_i \in \Re^{d\times d}$, $G_i \in \Re^{d\times m}$, $T_i \in \Re^{d\times n}$, $j_i \in \Re^{d\times 1}$, $p_i \in \Re^{m\times 1}$ and $z_i \in \Re^{d\times 1}$. Here,

d and q are the observer order and the number of robust observers, respectively.

Similarly, the fault detection filter equation may be written as

$$\dot{\hat{x}}(t) = A_n\hat{x}(t) + B_n u(t) + D(y(t) - C_n\hat{x}(t)), \tag{4.10}$$

$$\hat{y}(t) = C_n\hat{x}(t) \tag{4.11}$$

where $D \in \Re^{n\times m}$, $\hat{y} \in \Re^{m\times 1}$ and $\hat{x} \in \Re^{n\times 1}$.

Hence, under nominal conditions, the dynamics of $x_n(t)$, $z_i(t)$ and $\hat{x}(t)$, for $t_o = 0$, are given by

$$x_n(t) = e^{A_n t}x_n(0) + \int_0^t e^{A_n(t-s)}B_n u(s)\,ds, \tag{4.12}$$

$$z_{i_n}(t) = e^{F_i t}z_i(0) + \int_0^t e^{F_i(t-s)}[G_i C_n x_n(s) + T_i B_n u(s)]\,ds, \tag{4.13}$$

$$\hat{x}_n(t) = e^{(A_n - DC_n)t}\hat{x}(0) + \int_0^t e^{(A_n-DC_n)(t-s)}[DC_n x_n(s) + B_n u(s)]\,ds \tag{4.14}$$

where $x_n(0)$, $z_i(0)$, and $\hat{x}(0)$ are the initial condition vectors. z_{i_n} and $\hat{x}_n$ are the nominal solutions of z_i and $\hat{x}$.

4.3 Perturbed Dynamics

Following perturbation theory methodology one gets

$$x(t) = x_n(t) + \epsilon\,\delta x(t) + o(\epsilon^2) \tag{4.15}$$

where ϵ is a small positive perturbation parameter, $o(\epsilon^2)$ denotes higher order terms in ϵ, and δx is the first order perturbation in x.

Using earlier notation and (4.3), $x(t)$ can be written as

$$x(t) = e^{A_n t}x_n(0) + \int_0^t e^{A_n(t-s)}B_n u(s)\,ds + \epsilon \int_0^t e^{A_n(t-s)}[\delta A x(s) + \delta B u(s)]\,ds + o(\epsilon^2) \tag{4.16}$$

which can also be written as

$$x(t) = x_n(t) + \epsilon \int_0^t e^{A_n(t-s)}[\delta A[x_n(s) + \epsilon\,\delta x(s) + o(\epsilon^2)] + \delta B u(s)]\,ds + o(\epsilon^2) \tag{4.17}$$

Minor manipulation yields

$$x(t) = x_n(t) + \epsilon \int_0^t e^{A_n(t-s)}\delta A x_n(s)\,ds + \epsilon \int_0^t e^{A_n(t-s)}\delta B u(s)\,ds + o(\epsilon^2) \tag{4.18}$$

where $x_n(t)$ is the nominal solution given by (4.12).

For notational simplicity, it is convenient to write the above result as follows:

$$x(t) = x_n(t) + \epsilon\,\overline{\delta A}(t) + \epsilon\,\overline{\delta B}(t) + o(\epsilon^2) \tag{4.19}$$

where $\overline{\delta A}(t)$ and $\overline{\delta B}(t)$ are given by

$$\overline{\delta A}(t) = \int_0^t e^{A_n(t-s)}\delta A x_n(s)\,ds \tag{4.20}$$

$$\overline{\delta B}(t) = \int_0^t e^{A_n(t-s)}\delta B u(s)\,ds \tag{4.21}$$

Similarly, $y(t)$ can be written as

$$y(t) = y_n(t) + \epsilon\,\delta y(t) + o(\epsilon^2) \tag{4.22}$$

where $y_n(t)$ and $\delta y(t)$ are defined as

$$y_n(t) = C_n x_n(t), \tag{4.23}$$

$$\delta y(t) = C_n(\overline{\delta A}(t) + \overline{\delta B}(t)) + \delta C x_n(t) \tag{4.24}$$

4.4 ith Robust Observer Dynamics

This section is divided into three parts: in the first part, the state dynamics of the ith robust observer $z_i(t)$ is presented. In the second part, the dynamics of the robust observers error signal $e_i(t)$ is outlined. Finally, the dynamics are analyzed under parameter perturbations in δA,δB, and δC. For simplicity, this analysis will be carried out for constant nominal control input $u(t)$.

4.4.1 $z_i(t)$ Time Evolution

The time evolution of the ith robust observer state $z_i(t)$ will be carried for parameter perturbations in δA, δB, and δC. Integrating (4.8) leads to

$$z_i(t) = e^{F_i t} z_i(0) + \int_0^t e^{F_i(t-s)}[G_i y_n(s) + G_i \delta y(s) + T_i B_n u(s)]\, ds + o(\epsilon^2) \quad (4.25)$$

$$z_i(t) = z_{i_n}(t) + \int_0^t e^{F_i(t-s)} G_i[C_n(\overline{\delta A}(s) + \overline{\delta B}(s)) + \delta C x_n(s)]\, ds + o(\epsilon^2) \quad (4.26)$$

Which can be written as

$$z_i(t) = z_{i_n}(t) + \epsilon\, \delta z_i(t) + o(\epsilon^2) \quad (4.27)$$

where $z_{i_n}(t)$ is the nominal observer's state dynamics, and

$$\delta z_i(t) = \widehat{\delta A}(t) + \widehat{\delta B}(t) + \widehat{\delta C}(t) \quad (4.28)$$

with

$$\widehat{\delta A}(t) = \int_0^t e^{F_i(t-s)} G_i C_n \overline{\delta A}(s)\, ds, \quad (4.29)$$

$$\widehat{\delta B}(t) = \int_0^t e^{F_i(t-s)} G_i C_n \overline{\delta B}(s)\, ds, \quad (4.30)$$

$$\widehat{\delta C}(t) = \int_0^t e^{F_i(t-s)} G_i \delta C x_n(s)\, ds \quad (4.31)$$

4.4.2 $e_i(t)$ Solution Dynamics

The sensitivity of the robust observer error signal $e_i(t)$, to parameter perturbations in δA, δB, and δC will be addressed in this section. Moreover, an upper bound on the error dynamics $e_i(t)$ will be established. The time evolution of $e_i(t)$, under parameter perturbations, can be written as follows

$$e_i(t) = j_i^T(z_{i_n}(t) + \epsilon\,\delta z_i(t)) + p_i^T(y_n(t) + \epsilon\,\delta y(t)) + o(\epsilon^2) \tag{4.32}$$

Now, $e_i(t)$ can also be written as follows

$$e_i(t) = e_{i_n}(t) + \epsilon\,\delta e_i(t) + o(\epsilon^2) \tag{4.33}$$

where $e_{i_n}(t)$ and $\delta e_i(t)$ are given by

$$e_{i_n}(t) = j_i^T\delta z_i(t) + p_i^T\delta y(t) \tag{4.34}$$

$$\delta e_i(t) = j_i^T z_{i_n}(t) + p_i^T y_n(t) \tag{4.35}$$

Writing $\delta e_i(t)$ in terms of the perturbation matrices yields

$$\delta e_i(t) = [j_i^T\widehat{\delta A}(t) + p_i^T C_n\overline{\delta A}(t)] + [j_i^T\widehat{\delta B}(t) + p_i^T C_n\overline{\delta B}(t)] + [j_i^T\widehat{\delta C}(t) + p_i^T\delta C x_n(t)] \tag{4.36}$$

As discussed earlier, the binary decision logic, used to detect faults, is based on $e_i(t)$. Namely, whenever failure i occurs, the following inequality is verified

$$\| e_{i_n}(t) \| \geq \beta_t \tag{4.37}$$

and the complementary condition

$$\| e_{i_n}(t) \| < \beta_t \tag{4.38}$$

is true otherwise.

The threshold value β_t is selected during the design phase.

Nevertheless, when the system is subjected to either parameter perturbations, noise, modeling errors, or all of the above, then false alarms can be declared! That is, condition (4.37) can be activated in the absence of failures. This scenario can be translated into the following false alarms condition

$$\| e_{i_n}(t) + \epsilon\, \delta e_i(t) + o(\epsilon^2) \| \geq \beta_t,\ i = 1, 2, ..., q. \tag{4.39}$$

where q is the total number of robust observers.

In order to make use of this false alarm condition, equation (4.28) is written as follows

$$\| e_{i_n}(t) \| + | \epsilon | \| \delta e_i(t) \| + o(\epsilon^2) \geq \beta_t,\ i = 1, 2, ..., q \tag{4.40}$$

which is, obviously, a conservative false alarm condition, that is, when (4.40) holds, one concludes that there is a possibility of false alarms. The latter conservative false alarm condition, can be approximated, after neglecting $o(\epsilon^2)$ terms, and rewritten as

$$| \epsilon | \| \delta e_i(t) \| \geq \beta_t - \| e_{i_n}(t) \|,\ i = 1, 2, ..., q \tag{4.41}$$

Therefore, parameter perturbations for which (4.41) is satisfied produces a false alarm. In order to have a better understanding of this conservative false alarm condition (4.41), under parameter perturbations, one can compute an upper bound on $\| \delta e_i(t) \|$. First one writes

$$\| e_{i_n}(t) \| \leq \| j_i^T z_{i_n}(t) \| + \| p_i^T C_n x_n(t) \| \tag{4.42}$$

Similarly, it follows that

$$\| \delta e_i(t) \| \leq \| j_i^T \widehat{\delta A}(t) \| + \| p_i^T C_n \overline{\delta A}(t) \| + \| j_i^T \widehat{\delta B}(t) \| +$$

$$\| p_i^T C_n \overline{\delta B}(t) \| + \| j_i^T \widehat{\delta C}(t) \| + \| p_i^T \delta C x_n(t) \| \tag{4.43}$$

Now, substituting the bounds on $\| e_{i_n}(t) \|$ and $\| e_i(t) \|$ in (4.39) leads to the following inequality

$$\left\{ \| j_i^T z_{i_n}(t) \| + \| p_i^T C_n x_n(t) \| \right\} + | \epsilon | \left\{ \| j_i^T \widehat{\delta A}(t) \| + \| p_i^T C_n \overline{\delta A}(t) \| + \right.$$

$$\left. \| j_i^T \widehat{\delta B}(t) \| + \| p_i^T C_n \overline{\delta B}(t) \| + \| j_i^T \widehat{\delta C}(t) \| + \| p_i^T \delta C x_n(t) \| \right\} \leq \beta_t \tag{4.44}$$

$$\left\{ \| j_i^T \| \| z_{i_n}(t) \| + \| p_i^T C_n \| \| x_n(t) \| \right\} + | \epsilon | \left\{ \| j_i^T \| \| \widehat{\delta A}(t) \| + \right.$$

$$\| p_i^T C_n \| \| \overline{\delta A}(t) \| + \| j_i^T \| \| \widehat{\delta B}(t) \| + \| p_i^T C_n \| \| \overline{\delta B}(t) \| +$$

$$\left. \| j_i^T \| \| \widehat{\delta C}(t) \| + \| p_i^T \delta C \| \| x_n(t) \| \right\} \leq \beta_t \tag{4.45}$$

The above expressions are quite complex, and shed little light on the problem. Therefore, to circumvent this limitation, appropriate upper bounds on each quantity in (4.45) will be derived next.

To derive a simple upper bound on $\| x_n(t) \|$ one starts by the following step

$$\| x_n(t) \| = \| e^{A_n t} x_n(0) + \int_0^t e^{A_n (t-s)} B_n u(s)\, ds \| \tag{4.46}$$

After standard manipulation, one gets

$$\| x_n(t) \| \leq \| x_n(0) \| e^{\alpha(A_n)t} + \| B_n \| \int_0^t e^{\alpha(A_n)(t-s)} \| u(s) \| \, ds \tag{4.47}$$

where $\alpha(A)$ is called the logarithmic norm of A and it is defined as follows [17]

$$\alpha(A) \equiv max \left\{ \lambda(\frac{A^T + A}{2}) \right\} \tag{4.48}$$

Similarly, the upper bound on $\| z_{i_n}(t) \|$ is found to be

$$\| z_{i_n}(t) \| \leq \| z_i(0) \| e^{\alpha(F_i)t} + \| G_i C_n \| \int_0^t e^{\alpha(F_i)(t-s)} \| x_n(s) \| \, ds +$$

$$\| T_i B_n \| \int_0^t e^{\alpha(F_i)(t-s)} \| u(s) \| \, ds \tag{4.49}$$

Using the upper bound on $\| x_n(t) \|$, (4.49) can be rewritten as follows

$$\| z_{i_n}(t) \| \leq \| z_i(0) \| e^{\alpha(F_i)t} + \| G_i C_n \| \| x_n(0) \| \int_0^t e^{\alpha(F_i)(t-s)+\alpha(A_n)s} ds +$$

$$\| G_i C_n \| \| B_n \| \int_0^t \int_0^s e^{\alpha(F_i)(t-s)+\alpha(A_n)(s-w)} \| u(w) \| \, dw \, ds +$$

$$\| T_i B_n \| \int_0^t e^{\alpha(F_i)(t-s)} \| u(s) \| \, ds \tag{4.50}$$

Next, integrating the second term on the right-hand side of (4.50), leads to

$$\| z_{i_n}(t) \| \leq \| z_i(0) \| e^{\alpha(F_i)t} + \| G_i C_n \| \| x_n(0) \| te^{\alpha(A_n)t} +$$

$$\| G_i C_n \| \| B_n \| \int_0^t \int_0^s e^{\alpha(F_i)(t-s)+\alpha(A_n)(s-w)} \| u(w) \| \, dw \, ds +$$

$$\| T_i B_n \| \int_0^t e^{\alpha(F_i)(t-s)} \| u(s) \| \, ds \tag{4.51}$$

The upper bound on $\| \overline{\delta A}(t) \|$ is computed to be

$$\| \overline{\delta A}(t) \| \leq \| \delta A \| \int_0^t e^{\alpha(A_n)(t-s)} \| x_n(s) \| \, ds \tag{4.52}$$

The upper bound on $\| \overline{\delta B}(t) \|$ is computed to be

$$\| \overline{\delta B}(t) \| \leq \| \delta B \| \int_0^t e^{\alpha(A_n)(t-s)} \| u(s) \| \, ds \tag{4.53}$$

The upper bound on $\| \widehat{\delta A}(t) \|$ is found to be

$$\| \widehat{\delta A}(t) \| \leq \| G_i C_n \| \| \delta A \| \left\{ \| x_n(0) \| te^{\alpha(A_n)t} + \right.$$

$$\left. \| B_n \| \int_0^t \int_0^{s_1} \int_0^{s_2} e^{\alpha(A_n)(t-w)} \| u(w) \| \, dw \, ds_2 \, ds_1 \right\} \tag{4.54}$$

The upper bound on $\| \widehat{\delta B}(t) \|$ is computed to be

$$\| \widehat{\delta B}(t) \| \leq \| G_i C_n \| \| \delta B \| \int_0^t \int_0^s e^{\alpha(F_i)(t-s)+\alpha(A_n)(s-w)} \| u(w) \| \, dw \, ds \tag{4.55}$$

The upper bound on $\| \widehat{\delta C}(t) \|$ is found to be

$$\| \widehat{\delta C}(t) \| \leq \| G_i \delta C \| \left\{ \| x_n(0) \| te^{\alpha(A_n)t} + \right.$$

$$\left. \int_0^t \int_0^s e^{\alpha(F_i)(t-s)+\alpha(A_n)(s-w)} \| u(w) \| \, dw \, ds \right\} \tag{4.56}$$

Thus, after the norms of equation (4.45) have been computed, then these values are substituted into (4.45).

4.4.3 Analysis of the Solution

Now, one can utilize these upper bounds, to find which robust observer is more sensitive to parameter perturbations in δA, δB and δC then the others. Thus, the norm of the quantities in (4.45) will be computed for a constant nominal input $u(t) = u_n$, for each robust observer as follows:

1. Evaluate the upper bound on $\| x_n(t) \|$ and for $u = u_n$, to get

$$\| x_n(t) \| \leq \| x_n(0) \| e^{\alpha(A_n)t} + \| B_n \| \| u \| \int_0^t e^{\alpha(A_n)(t-s)} ds \tag{4.57}$$

Let the norm of u, be $\| u \| = | u_n |$. Thus, integrating the previous equation, will lead to

$$\| x_n(t) \| \leq [\| x_n(0) \| + \| B_n \| | u_n | t] e^{\alpha(A_n)t} \tag{4.58}$$

2. The upper bound on $\| z_{i_n}(t) \|$ is found to be

$$\| z_{i_n}(t) \| \leq \left[\| G_i C_n \| \| x_n(0) \| + \alpha(A_n)^{-1} \| G_i C_n \| \| B_n \| | u_n | \right] te^{\alpha(A_n)t} +$$

$$\left[\| z_i(0) \| + \| T_i B_n \| | u_n | t - \alpha(A_n)^{-1} \| G_i C_n \| \| B_n \| | u_n | \right] e^{\alpha(F_i)t} \tag{4.59}$$

3. The upper bound on $\| \overline{\delta A}(t) \|$ is computed as

$$\| \overline{\delta A}(t) \| \leq \| \delta A \| \| x_n(0) \| e^{\alpha(A_n)t} \tag{4.60}$$

4. The upper bound on $\| \overline{\delta B}(t) \|$ is found to be

$$\| \overline{\delta B}(t) \| \leq \| \delta B \| | u_n | te^{\alpha(A_n)t} \tag{4.61}$$

5. The upper bound on $\| \widehat{\delta A}(t) \|$ is found to be

$$\| \widehat{\delta A}(t) \| \leq \| G_i C_n \| \| \delta A \| \left\{ \| x_n(0) \| + \| B_n \| | u_n | \frac{t}{2\alpha(A_n)} \right\} te^{\alpha(A_n)t} \tag{4.62}$$

6. The upper bound on $\| \widehat{\delta B}(t) \|$ is computed as

$$\| \widehat{\delta B}(t) \| \leq \| G_i C_n \| \| \delta B \| | u_n | \alpha(A_n)^{-1} t \left[e^{\alpha(A_n)t} - e^{\alpha(F_i)t} \right] \tag{4.63}$$

7. Finally, the upper bound on $\| \widehat{\delta C}(t) \|$ is

$$\| \widehat{\delta C}(t) \| \leq \| G_i \| \| \delta C \| \left\{ \left[\| x_n(0) \| + \alpha(A_n)^{-1} \mid u_n \| \right] te^{\alpha(A_n)t} - \right.$$

$$\left. \alpha(A_n)^{-1} \mid u_n \mid e^{\alpha(F_i)t} \right\} \tag{4.64}$$

Hence, the effect of perturbations δA, δB, or δC, on the robust observer error signal $e_i(t)$, can be analyzed more easily. Consequently, false alarms which are mainly caused by the term $\delta e_i(t)$, will be analyzed next. Thus, the upper bound on $\| \delta e_i(t) \|$ can be written as follows

$$\| \delta e_i(t) \| \leq \| j_i^T \| \| \widehat{\delta A}(t) \| + \| p_i^T C_n \| \| \overline{\delta A}(t) \| + \| j_i^T \| \| \widehat{\delta B}(t) \| +$$

$$\| p_i^T C_n \| \| \overline{\delta B}(t) \| + \| j_i^T \| \| \widehat{\delta C}(t) \| + \| p_i^T \| \| \delta C \| \| x_n(t) \| \tag{4.65}$$

Next, substituting the norms of individual quantities in (4.65), to obtain

$$\| \delta e_i(t) \| \leq \left\{ \| j_i^T \| \| x_n(0) \| + \| p_i^T C_n \| \| G_i C_n \| \left[\| x_n(0) \| + \right. \right.$$

$$\left. \left. \| B_n \| \mid u_n \mid \frac{t}{2\,\alpha(A_n)} \right] t \right\} e^{\alpha(A_n)t} \| \delta A \| +$$

$$\left\{ \| j_i^T \| \| G_i C_n \| \alpha(A_n)^{-1} t [e^{\alpha(A_n)t} - e^{\alpha(F_i)t}] + \| p_i^T C_n \| e^{\alpha(A_n)t} \right\} \mid u_n \mid \| \delta B \| \; t +$$

$$\left\{ \| j_i^T \| \| G_i \| \left\{ \| x_n(0) \| + \alpha(A_n)^{-1} \mid u_n \| \right] te^{\alpha(A_n)t} - \alpha(A_n)^{-1} \mid u_n \mid e^{\alpha(F_i)t} \right\} +$$

$$\| p_i^T \| \left[\| x_n(0) \| + \| B_n \| \mid u_n \mid t \right] e^{\alpha(A_n)t} \Big\} \| \delta C \| \tag{4.66}$$

The previous equation (4.66) provides a quantitative result for the sensitivity of the ith observer error signal to parameter perturbation. Equation (4.66) shows that the term δe_i is more sensitive to any level of parameter perturbation in δA, δB,

or δC. It also shows that the design parameters of the observer $(F_i, p_i, G_i, T_i, j_i)$ have different influences upon the observer error. For example, in order to speed up the response of the observer, the poles of the ith observer have to be selected to lie to the left of the system eigenvalues in the complex plane and this implies that G_i has to be large. However, when G_i is large, then the observer error will be more sensitive to parameter perturbations, noise and modeling errors (i.e. any small perturbation will be magnified into the observer error). Thus, it is desirable to find a compromise value of G_i that will satisfy both the stability and the minimal sensitivity to parameter perturbation requirements. This observation can be seen by computing the norm of G_i as follows

$$\| G_i \| = \| (T_i A_n - F_i T_i) C^T (CC^T)^{-1} \| \tag{4.67}$$

After some manipulation, the upper bound on $\| G_i \|$ reduces to

$$\| G_i \| \leq \| T_i \| \ (\| A_n + \| F_i \|) \ \| C^T (CC^T)^{-1} \| \tag{4.68}$$

but since F_i is in Jordan canonical form, then

$$F_i = V_i + N_i \tag{4.69}$$

where V_i is a diagonal matrix, and N_i is the nilpotent matrix which consists of ones above the main diagonal and the rest of the elements are zeros.

Thus, the upper bound on $\| F_i \|$ is

$$\| F_i \| \leq \| V_i \| + \| N_i \| \tag{4.70}$$

Since, the bound on $\| N_i \|$ is

$$\| N_i \| = [\lambda(N_i^T N_i)]^{1/2} = [\lambda_{max}(diag([0, 1, 1, ..., 1])]^{1/2} = 1 \tag{4.71}$$

Also, the matrix V_i consists of the observer eigenvalues, which are selected to lie to the left of the system eigenvalues (i.e maximum eigenvalue of $(A_n^* A_n)$). Now, let us select the observer eigenvalue to be

$$\mu = \beta \lambda_{m_1} \tag{4.72}$$

where $\lambda_{m_1} = [\lambda_{max}(A_n^* A_n)]^{1/2}$, and β is a scalar value ($\beta \geq 1$).

Thus, the norm of A_n and V_i is as follows

$$\| A_n \| = \lambda_{m_1} = [\lambda_{max}(A_n^* A_n)]^{1/2} \tag{4.73}$$

$$\| V_i \| = [\lambda_{max}(diag([\mu^2, \mu^2, ..., \mu^2])]^{1/2} = \mu = \beta \lambda_{m_1} \tag{4.74}$$

Then, substituting the norms of A_n, V_i, and N_i into the norm of G_i, which leads to

$$\| G_i \| \leq ((1+\beta) \lambda_{m_1} + 1) \| T_i \| \| C^T (C C^T)^{-1} \| \tag{4.75}$$

Hence, the above upper bound on G_i shows that as the observer pole μ is selected to lie to the left of the maximum eigenvalue of the system, then the observer gain G_i will also increase (i.e. β increases).

4.5 $\hat{x}(t)$ Time Evolution

The time evolution solution of the detection filter state equation (4.10) will be divided into three parts: in the first part the solution of $\hat{x}(t)$ is presented. In the second part the dynamics of residual signal $\bar{e}(t)$ is given. Finally, the solution is analyzed under parameter perturbations in δA, δB, and δC. This analysis will be carried out for constant nominal control input u.

4.5.1 Perturbed Dynamics

The solution of the detection filter equation under parameter perturbations in δA, δB, and δC will be presented in this section. With the aid of (4.10), on integrating this equation, will yield

$$\hat{x}(t) = e^{(A_n - DC_n)t}\hat{x}(0) + \int_0^t e^{(A_n - DC_n)(t-s)}[Dy(s) + B_n u(s)]\, ds, \quad (4.76)$$

$$\hat{x}(t) = \hat{x}_n(t) + \epsilon \int_0^t e^{(A_n - DC_n)(t-s)} D\delta y(s)\, ds + o(\epsilon^2) \quad (4.77)$$

where $\hat{x}_n(t)$ is the nominal solution of $\hat{x}(t)$.

$$\hat{x}_n(t) = e^{(A_n - DC_n)t}\hat{x}(0) + \int_0^t e^{(A_n - DC_n)(t-s)}[Dy_n(s) + B_n u(s)]\, ds \quad (4.78)$$

Next, equation (4.78) can be written as follows

$$\hat{x}(t) = \hat{x}_n(t) + \epsilon\, \widehat{\delta x}(t) + o(\epsilon^2) \quad (4.79)$$

where $\hat{x}_n(t)$ is defined by (4.14), and $\widehat{\delta x}(t)$ is given as follows

$$\widehat{\delta x}(t) = \widehat{\delta A}(t) + \widehat{\delta B}(t) + \widehat{\delta C}(t) \quad (4.80)$$

where $\widehat{\delta A}(t)$, $\widehat{\delta B}(t)$, and $\widehat{\delta C}(t)$ are

$$\widehat{\delta A}(t) = \int_0^t e^{(A_n - DC_n)(t-s)} DC_n \overline{\delta A}(s)\, ds, \quad (4.81)$$

$$\widehat{\delta B}(t) = \int_0^t e^{(A_n - DC_n)(t-s)} DC_n \overline{\delta B}(s)\, ds, \quad (4.82)$$

$$\widehat{\delta C}(t) = \int_0^t e^{(A_n - DC_n)(t-s)} D\delta C x_n(s)\, ds \quad (4.83)$$

Next, writing (4.80) in terms of the perturbation matrices, leads to

$$\widehat{\delta x}(t) = \int_0^t e^{(A_n - DC_n)(t-s)} DC_n \int_0^s e^{A_n(s-w)} \delta A x_n(w)\, dw\, ds +$$

$$\int_0^t e^{(A_n - DC_n)(t-s)} DC_n \int_0^s e^{A_n(s-w)} \delta B u(w)\, dw\, ds + \int_0^t e^{(A_n - DC_n)(t-s)} D\delta C x_n(s)\, ds \tag{4.84}$$

4.5.2 $\overline{e}(t)$ solution Dynamics

Now defining the error between the detection filter and the system to be as follows:

$$\overline{e}(t) = J(C_n \hat{x}(t) - y(t)) \tag{4.85}$$

Using (4.22) and (4.79) in the previous equation leads to

$$\overline{e}(t) = \overline{e}_n(t) + \epsilon\, \overline{\delta e}(t) + o(\epsilon^2) \tag{4.86}$$

where $\overline{e}_n(t)$ and $\overline{\delta e}(t)$ are

$$\overline{e}_n(t) = JC_n(\hat{x}_n(t) - x_n(t)), \tag{4.87}$$

$$\overline{\delta e}(t) = J\left[C_n(\widehat{\delta A}(t) - \overline{\delta A}(t)) + C_n(\widehat{\delta B}(t) - \overline{\delta B}(t)) + C_n\, \widehat{\delta C}(t) - \delta C x_n(t)\right] \tag{4.88}$$

Equation (4.85) can also be written as

$$\overline{e}(t) = JC_n\left[\hat{x}_n(t) - x_n(t)\right] + \epsilon \left\{ JC_n\left[(\widehat{\delta A}(t) - \overline{\delta A}(t)) + (\widehat{\delta B}(t) - \overline{\delta B}(t))\right] + J(C_n\, \widehat{\delta C}(t) - \delta C x_n(t))\right\} + o(\epsilon^2) \tag{4.89}$$

Thus, the term $\overline{\delta e}(t)$ in (4.86) models the effect of parameter perturbation in the system. Hence, it is this term that causes false alarms in the binary decision

logic (i.e. once an alarm signal is declared, it is not possible to attribute this to a real fault in the system or due to parameter perturbation). Furthermore, in order to have a false alarm signal, the following two conditions have to be satisfied.

1. The error $\| \bar{e}(t) \| \geq \beta_t$, where β_t is a threshold value.

2. The angles $\theta_i \leq \theta_t$, for $i = 1, 2, ..., n$, where θ_t is a threshold angle, and θ_i's are computed from (2.24).

The threshold values β_t and θ_t are selected during detection filter design phase. Thus, computing $\| \bar{e}(t) \|$ leads to

$$\| \bar{e}(t) \| = \| \bar{e}(t) + \epsilon \overline{\delta e}(t) + o(\epsilon^2) \| \tag{4.90}$$

In order to utilize the above equation, it is rewritten as

$$\| \bar{e}(t) \| \leq \| \bar{e}_n(t) \| + | \epsilon | \| \overline{\delta e}(t) \| + o(\epsilon^2) \tag{4.91}$$

where $\| \bar{e}(t) \|$ is the upper bound on $\bar{e}(t)$. $\| \bar{e}_n(t) \|$ and $\| \overline{\delta e}(t) \|$ are given by

$$\| \bar{e}_n(t) \| \leq \| JC_n \| \| \hat{x}_n(t) - x_n(t) \| \tag{4.92}$$

$$\| \dot{\overline{\delta e}}(t) \| \leq \left\{ \| JC_n \| \left[\| \widehat{\overline{\delta A}}(t) \| + \| \overline{\delta A}(t) \| + \| \widehat{\overline{\delta B}}(t) \| + \| \overline{\delta B}(t) \| \right] + \right.$$
$$\left. \| J \| \left[\| C_n \| \| \widehat{\overline{\delta C}}(t) \| + \| \delta C \| \| x_n(t) \| \right] \right\} \tag{4.93}$$

In order to have a better understanding of conditions (4.92) and (4.93) under parameter perturbations, one can compute an upper bound on $\| \overline{\delta e}(t) \|$. The upper bounds on $\| x_n(t) \|$, $\| \overline{\delta A}(t) \|$, and $\| \overline{\delta B}(t) \|$ have been computed in section 4.4. Thus, in this section, the upper bounds on $\| \hat{x}_n(t) - x_n(t) \|$, $\| \widehat{\overline{\delta A}}(t) \|$, $\| \widehat{\overline{\delta B}}(t) \|$, and $\| \widehat{\overline{\delta C}}(t) \|$ will be found.

To derive a simple upper bound on $\| \hat{x}_n(t) - x_n(t) \|$, one starts by the following step

$$\| \hat{x}_n(t) - x_n(t) \| = \| \left[e^{(A_n - DC_n)t} \hat{x}(0) - e^{A_n t} x_n(0) \right] + \int_0^t \left[e^{(A_n - DC_n)(t-s)} - e^{A_n(t-s)} \right] B_n u(s)\, ds + \int_0^t e^{(A_n - DC_n)(t-s)} DC_n x_n(s)\, ds \| \tag{4.94}$$

After some manipulation, one gets

$$\| \hat{x}_n(t) - x_n(t) \| \leq \| DC_n \| \left\{ \| x_n(0) \left(1 + e^{\|DC_n\| t}\right) t e^{\alpha(A_n)t} + \| B_n \| \int_0^t (t-s) e^{(\alpha(A_n) + \|DC_n\|)(t-s)} \| u(s) \|\, ds + \| B_n \| \int_0^t \int_0^s e^{\alpha(A_n - DC_n)(t-s) + \alpha(A_n)(s-w)} \| u(w) \|\, dw\, ds \right\} \tag{4.95}$$

The upper bound on $\| \widehat{\delta A}(t) \|$ is

$$\| \widehat{\delta A}(t) \| \leq \| DC_n \| \| \delta A \| \| x_n(0) \|\, t e^{\alpha(A_n - DC_n)t} + \| DC_n \| \| B_n \| \| \delta A \| \int_0^t \int_0^z \int_0^r e^{\alpha(A_n - DC_n)(t-z) + \alpha(A_n)(z-w)} \| u(w) \|\, dw\, dr\, dz \tag{4.96}$$

The upper bound on $\| \widehat{\delta B}(t) \|$ is

$$\| \widehat{\delta B}(t) \| \leq \| DC_n \| \| \delta B \| \int_0^t \int_0^s e^{\alpha(A_n - DC_n)(t-s) + \alpha(A_n)(s-w)} \| u(w) \|\, dw\, ds \tag{4.97}$$

Finally, the upper bound on $\| \widehat{\delta C}(t) \|$ is

$$\| \widehat{\delta C}(t) \| \leq \| D\delta C \| \| x_n(0) \|\, t e^{\alpha(A_n)t} + \| D\delta C \| \| B_n \| \int_0^t \int_0^s e^{\alpha(A_n - DC_n)(t-s) + \alpha(A_n)(s-w)} \| u(w) \|\, dw\, ds \tag{4.98}$$

After evaluating the upper bounds on the quantities in (4.92) and (4.93), one can study the effect of parameter perturbations on $\overline{\delta e_i}(t)$ more easily.

4.5.3 Solution Analysis

In this part, the solution of the detection filter error equation will be analyzed for a constant nominal input $u(t)$, i.e. $u(t) = u_n$ as follows

1. With the aid of (4.95), evaluating the upper bound on $parallel\hat{x}_n(t) - x_n(t) \parallel$, will lead to

$$\| \hat{x}_n(t) - x_n(t) \| \leq \| DC_n \| \left\{ \| x_n(0) \| (1 + e^{\|DC_n\|t}) te^{\alpha(A_n)t} + \right.$$
$$\left. | u_n | \| B_n \| \left[\alpha(A_n - DC_n)^{-1}(e^{\alpha(A_n - DC_n)t} - 1) + \frac{(e^{\alpha(A_n - DC_n)t} - e^{\alpha(A_n)t})}{\alpha(A_n) - \alpha(A_n - DC_n)} \right] \right\} \tag{4.99}$$

2. The upper bound on $\| \widehat{\delta A}(t) \|$ is

$$\| \widehat{\delta A}(t) \| \leq \| DC_n \| \| \delta A \| \| x_n(0) \| te^{\alpha(A_n)t} \tag{4.100}$$

3. The upper bound on $\| \widehat{\delta B}(t) \|$ is

$$\| \widehat{\delta B}(t) \| \leq \| DC_n \| \| \delta B \| | u_n | \alpha(A_n)^{-1} t(e^{\alpha(A_n)t} - e^{\alpha(A_n - DC_n)t}) \tag{4.101}$$

4. The upper bound on $\| \widehat{\delta C}(t) \|$ is

$$\| \widehat{\delta C}(t) \| \leq \left\{ \| x_n(0) \| te^{\alpha(A_n)t} + \right.$$
$$\left. \| B_n \| | u_n | \alpha(A_n)^{-1} t(e^{\alpha(A_n)t} - e^{\alpha(A_n - DC_n)t} \right\} \| D \| \| \delta C \| \tag{4.102}$$

Hence, after the individual upper bounds on (4.92) and (4.93) have been computed, then one can study the effect of perturbations (δA, δB, and δC) on the detection filter more easily. Thus, the upper bound on $\| \overline{\delta e}(t) \|$ is

$$\| \overline{\delta e}(t) \| \leq \| JC_n \| \| x_n(0) \| [\| DC_n \| + 1] te^{\alpha(A_n)t} \| \delta A \| +$$

$$\| u_n \| \| JC_n \| \left[\| DC_n \| \frac{(e^{\alpha(A_n)t} - e^{\alpha(A_n - DC_n)t})}{\alpha(A_n)} + e^{\alpha(A_n)t} \right] \| \delta B \| t +$$
$$\| J \| \left\{ \| C_n \| \left[\| x_n(0) \| e^{\alpha(A_n)t} + \| B_n \| \| u_n \| \frac{(e^{(\alpha(A_n)t} - e^{\alpha(A_n - DC_n)t})}{\alpha(A_n)} \right] \| D \| \right.$$
$$\left. + [\| x_n(0) \| + \| B_n \| \| u_n \| t] e^{\alpha(A_n)t} \right\} \| \delta C \| \qquad (4.103)$$

Equation (4.103) provides a quantitative result for the sensitivity of the detection filter to parameter perturbations. This equation shows that the term $\overline{\delta e}(t)$ is more sensitive to any level of parameter perturbations in δA, δB, and δC. It also shows that the gain D of the detection filter has different influences upon the filter error. For example, in order to make the detection filter stable, the poles of the filter have to be selected to lie to the left of the system eigenvalues and this implies that D has to be large. In addition, such a choice of D will cause the error $\overline{e}(t)$ to die out fast. However, when D is large, the detection filter will be more sensitive to parameter perturbations, noise, or modeling errors (i.e. any small perturbation in δA, δB, or δC will be magnified into $\overline{e}(t)$). Thus, it is desirable to select a value of D that will meet the following two requirements: stability of the detection filter and minimal sensitivity of the detection filter to parameter perturbations.

4.6 Discussion of a Case Study

One application of the sensitivity analysis that was presented in this chapter, is to study the sensitivity of the diagnosis scheme for the four-tank non-linear system

to a perturbation δ in the input. This analysis is very important to determine the acceptable range of performance for the robust observers and the detection filter. Table 4.1 summarizes the results. The range M for both the robust observers (M_R) and detection filter (M_D) are

$$M_R = \{-0.10 \leq \delta \leq 0.09\}, \tag{4.104}$$

$$M_D = \{-0.03 \leq \delta \leq 0.02\} \tag{4.105}$$

where the lower and upper bounds of M_R are chosen to be the maximum of δ_{min} and the minimum δ_{max} respectively.

The results outlined in (4.104) and (4.105), show that if the four-tank system is perturbed within a certain δ then false alarms will occur. These results are approximate because of the following:

1. Discrepancies between the linear and the non-linear models.

2. The derivation of perturbation equations in Chapter four were based on small parameter perturbations. Thus, if the perturbations occurring in the dynamical system are large then this analysis is not feasible.

Table 4.1: The limits of parameter perturbations for the observer-based schemes

Robust Observers		
i	δ_{min}	δ_{max}
1	-0.105	0.084
2	-0.254	0.515
3	-0.500	49.000
4	-0.810	-1.290
Detection Filter		
i	θ_{min}	θ_{max}
θ_1	-13.6	13.6
θ_2	79.2	100.8
θ_3	79.2	100.8
θ_4	82.9	95.1

Chapter 5

ROBUSTIFICATION OF FAILURE DETECTION AND ISOLATION ALGORITHMS VIA NEURAL NETWORKS

5.1 Introduction

One technique used to detect, and isolate failures, is the observer-based approach. This approach can be applied adequately, only if the system is correctly described by a linear dynamical model without any parameter perturbations. This is practically unfeasible, and parameter perturbations are always present due to several reasons, such as modeling errors. Consequently, there is always a discrepancy between the actual system parameters, and those of the design model.

Designing observers without taking into consideration possible uncertainties, will lead to state estimates containing deviations, which in turn will generate false alarms. One way to overcome this problem is to design observers which are insensitive to parameter perturbations. However, the design procedure for this class of observers can be quite complex [12].

A new approach that has been used to detect failures, is the Neural Network (NN) approach. Due to the many advantages offered by NNs , namely, robustness to noise and "generalization" to new patterns, the NN approach is robust to system non-linearities, modeling errors, noise and parameter perturbations. However, as it is shown in this work, the NN technique is not appropriate for slow systems, such as, the four-tank system used in this thesis. This is due to two reasons: (i) very long fault detection time (ii) extensive training time.

The fault detection time, i.e. time between the occurrence of a fault and its detection, is very crucial for some dynamical systems; for example, a failure in the cooling system of an exothermic chemical reactor will cause the outlet temperature to increase beyond the designed value, which must be detected almost instantly; otherwise, it will cause a blast in the reactor. Furthermore, the training time for slow systems, is very long due to the slow dynamics. Consequently, NNs approach is not efficient in detecting failures for this class of systems. As it will be demonstrated in this chapter, using the observer-based technique to detect failures in slow systems, is superior to the NN approach. But as it was shown in Chapter four, the observer-based approach, suffers from sensitivity problems.

In order to alleviate these problems, a solution which utilizes NN is proposed in this thesis. It basically combines the observer-based technique along with a NN. In this proposed approach, the states are estimated by the observer(s) and then the failure detection logic is implemented by the NN. Consequently, and as it will be shown in the sequel, this proposed FDI scheme will inherent the advantages of the observer-based technique and the NN, and hopefully minimize their disadvantages, such as sensitivity in the case of the observer-based technique and slowness for NNs.

Wile hoping to minimize the disadvantages of the observer-based and the NN's FDI techniques when used separately, by combining them, one is generating a number of interesting and important questions which will play a key role in the success of the proposed scheme. There are basically two conflicting issues. First, the amount of temporal information presented to the NN decision logic, and second, the complexity of the NN (i.e., number of processing elements). The simplest NN decision logic will result from feeding only the present residuals to the NN (i.e., $e_i(t),\ i = 1, 2, ..,)$. This will result in a memoryless decision logic, which could result in low performance. However, it is intuitively appealing to add memory to the decision logic by feeding lagged residuals. Nevertheless, the complexity of the NN will increase with the lag introduced to increase the temporal information. Therefore, it is obvious that a balance has to be obtained by compromizing between these two important design issues. As it will be shown, this design dichotomy will be of most importance in this work, especially when sensitivity and therefore,

false alarms are into question, along with training and validation times as well as implementation of the Neuronic Decision Logic.

This chapter is organized as follows. In section two a short introduction about NNs is given along with the back-propagation algorithm, and NNs topologies. In section three, the proposed new technique is introduced. Section four describes a complete case study. Section five concludes the chapter.

5.2 A Glimpse to Neural Networks (NNs)

NNs are dynamical systems composed of highly interconnected layers of simple neuron-like processing units (figures 5.1 and 5.2). These layers are classified as input layer, hidden layer (or layers), and output layer. In the input layer, the patterns are presented to the networks. In the output layer, the response of the NN is generated. For each connection between these layers, there is a certain weight associated with it.

NN operations consist of two main phases: a training phase, and a validation phase. In the training phase, the network is repeatedly presented with a set of random input-output patterns. In this thesis, learning is accomplished by the classical back-propagation algorithm [14]. In the second phase, a new set of random input-output patterns are presented to the networks. The objective of this test is to check the ability of the NN to classify patterns outside the training set.

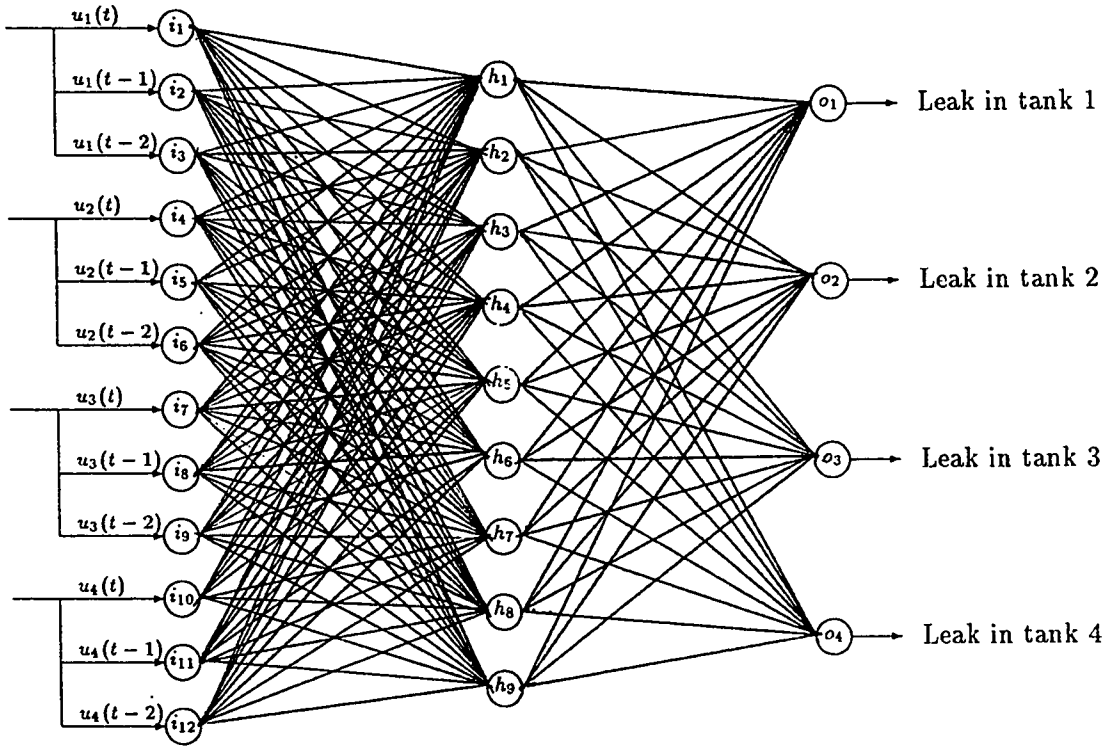


Figure 5.1: Moving window neural networks

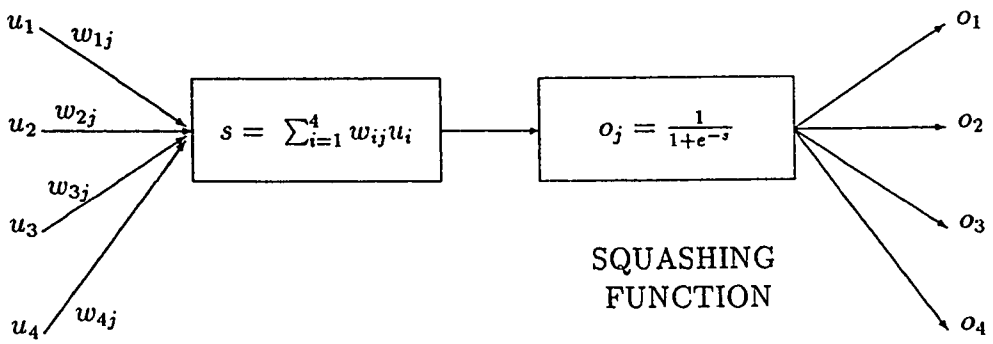


Figure 5.2: Simple neuron transfer function

NN operations are collectively performed in parallel computational fashion with the knowledge represented by the weights between the processing elements. This architecture enables NN to solve complex problems rapidly Rumelhart et al. [14].

5.2.1 The Back-Propagation Algorithm

The back-propagation algorithm (BA) also known as the delta rule [14] has been widely used in recent years and has produced successful learning applications in various fields, such as, in pattern recognition and in text-to-speech conversion. The BA is an error-correcting learning procedure, which was developed for training a multilayer feedforward NNs.

Feedforward NNs possess three layers: input, hidden and output layers. The input layer stores the input value while the hidden and output layers carry out two calculations each. First, a neuron in these two layers calculates a weighted sum of its inputs from the preceding layer as

$$s_j = \sum_{i=1}^{N} w_{ij} o_i \tag{5.1}$$

where w_{ij} is the connection weight from the ith node in the preceding layer to the jth node in the current layer, and o_i is the output from the ith node in the preceding layer. Note that for the input layer only, the output node for an input u_i is computed as

$$o_i = \frac{1}{1 + e^{-u_i}} \tag{5.2}$$

Second, the output of the neuron is calculated as the sigma function of the sum

$$o_j = f(s_j) = \frac{1}{1 + e^{-s_j}} \tag{5.3}$$

In the learning phase, the weights in the network are adjusted to minimize a sum of squared errors between predicted and actual outputs as

$$E = \sum_{i=1}^{N_p} E_i = \sum_{p=1}^{N_p} \sum_{j=1}^{N_L} \{t_{pj} - o_{pj}\}^2 \tag{5.4}$$

where N_p is the total number of training patterns, N_L is the total number of output nodes, and E_i represents the error on pattern i. The variables t_{pj} and o_{pj} are the desired target value and actual output for the jth output unit when pattern p has been presented.

The weights are adjusted in proportion to the gradient of the s quared error E. The new weights are computed as follows:

$$w_{ij}(n+1) = w_{ij}(n) + \zeta \Delta w_{ij}(n) + \eta \Delta w_{ij}(r-1) \tag{5.5}$$

where ζ and η are the learning rate and the momentum rate constants respectively. $\Delta w_{ij}(n)$ is calculated as follows

$$\Delta w_{ij}(n) = \sum_{p=1}^{N_p} \Delta w_{ij}^p(n) \tag{5.6}$$

where $\Delta w_{ij}^p(n)$ is the weight change with respect to the pth pattern. $\Delta w_{ij}^p(n)$ is proportional to the gradient of the error with respect to the weights, and it is computed as follows

(i) hidden to output weights

$$\Delta w_{ij}^p(n) = \frac{df(s_k)}{ds_k}[t_k(n) - o_k(n)]o_j(n) \tag{5.7}$$

(ii) input to hidden weights

$$\Delta w_{ij}^{p}(n) = \frac{df(s_j)}{ds_j}\left[\sum_{k=1}^{N_L}\frac{df(s_k)}{ds_k}[t_k(n) - o_k(n)]w_{jk}(n)\right] o_i(n) \qquad (5.8)$$

and

$$\frac{df(s_k)}{ds_k} = f(s_k)(1 - f(s_k)) \qquad (5.9)$$

After presentation of all input-output patterns, then the weights are changed with $\Delta w_{ij}(n)$. Usually, a non-linear activation function (eq. (5.3)) is used in each neuron due to two reasons: (i) existence of derivative, and (ii) non-linearity.

5.2.2 Neural Network Topology

The NN topology is the process that describes the general layout of the NN. Several NN topologies have been suggested in the literature [15]. Each differs in the number of processing nodes, the connections, the training procedures and how input-output patterns are processed. The most commonly use topology is a three layer feedforward NN [14]. The input layer receives m signals from the outside environment. These signals may undergo scaling operation so that they range between $[-1, 1]$ before processing by the NN. The hidden layer receives inputs from the input layer and then sends them to the output layer. The output layer gives the response of the NN to a particular set of inputs.

In the following section, two types of NN will be examined: the first NNs, called the naive neural network, processes current inputs only $u_i(t)$, $i = 1, ..., k$, and the second NN, called moving window neural network, uses current inputs as well as lagged (delayed) inputs i.e., $u_i(t)$, $u_i(t-1)$, $u_i(t-2)$. The objective of these lagged inputs is to study the effect of including lagged inputs on the performance of the NN.

5.3 The Observer/NNs-based proposed scheme

5.3.1 Motivation

As explained earlier, the observer-based FDI schemes are quite sensitive to parameter perturbations making them less reliable in face of unavoidable situation. Indeed, only parameter perturbation, might activate false alarms. This lack of robustness to parameter perturbations can be easily interpreted as a reduction in the threshold values designed as a key part of the failure detection scheme. This threshold reduction is "proportional" to the parameter perturbations induced variations. In other words, the more important the parameter variations, the greater the reduction in the threshold value, and consequently, the more frequent are the false alarms. As a matter of fact, as a limiting scenario, the threshold values can be reduced to zero, due to parameter variations, thus leading to a permanent false alarm situation.

Since the design of FDI filters, is quite simple, then it is important we keep

this advantage, and try to eliminate, or at least minimize the sensitivity of such technique. However, since the less sensitive observers are quite involved, design wise, it is of interest to keep the usual observer and try to enhance the Decision logic, by making it more tolerant to parameter perturbations induced variations. This is feasible via Neural Networks (NNs).

As a matter of fact, NN's are well known for their robustness to a wide class of perturbations and/or disturbance. For instance, NNs are known to work quite satisfactorily in a noisy environment. Moreover, and most importantly, NN's are known to correctly classify "unknown" patterns, whenever these patterns are close enough to the training patterns, this is roughly known as the "generalization" property of NN's.

Therefore, one hopes that by replacing the decision logic by a NN, and by correctly training it, this new system will be quite insensitive to perturbations and/or disturbances, especially, in our case, parameter perturbations (Fig. 5.3). That is, the main assumption made concerning this new scheme, is that mild parameter perturbations, will induce mild perturbations in the information processed by the NN, thus leading to less false alarms, by virture of the generalization property, and some level of robustification will be built in our FDI system.

The main question that imposes itself at this point is how "mild" is mild? In addition, does this robustified scheme conflict with other benefits that the unrobustified technique had? The two important questions, as well as others, will be

discussed, and partially answered in the sequel.

The previous discussion concentrated mainly on the robustification of the observer-based FDI scheme by implementing the required decision logic via NN's. Nevertheless, there is a second benefit to this marriage between these two techniques. As it was mentioned earlier, NN's have been used as FDI tools, but they turned out to be quite inappropriate when the dynamics to be monitored are quite slow. So, by using same observer-based FDI techniques along with NN's, one can considerably reduce this limitation. Unfortunately, this only time, when the dynamics under consideration have an adequate model, for when the model is not available, this hybrid combination is no longer possible.

5.3.2 Proposed Network Topologies

There are two major concerns when designing this new scheme: (i) temporal information (i.e., memory), (ii) complexity (i.e., size). Since the system is dealing with temporal patterns, it seems important to give it some memory. Unfortunately, and as it will be seen in the sequel, the more memory one requires, the more complex the system gets. So it is of prime importance to strike a balance between these two important and yet conflicting issues.

To deal with this problem, two classes of topologies are proposed, the first one has no memory, and is called the Naive Neural Network (NNN), the second one has a number of lagged residual information inputs, where this lag extend in the past

to a specific depth, and is called the Moving Window Neural Network (MWNN). These two topologies are discussed in more details in the sequel.

The naive neural network (NNN) is a three layer NN. The input layer processes inputs that depend on current time value only. The advantages of NNN are reasonable robustness to noise and parameter perturbations, training time is short, and the neural network topology is small compared to the moving window NN. One disadvantage of NNN is that it is has no memory.

The moving window neural network (MWNN) is a three layer NN. The input layer processes inputs that depend on current inputs and lagged inputs. A typical MWNN is shown in figure 5.1. In order to train this NN, the training data has to be sampled in a "moving window" fashion. This means that the input-output training data is sampled at time t (with the lagged inputs). At time $t+1$ the process is repeated in this manner until the whole data is sampled. This is in contrast to NNN where the inputs are sampled only at current time.

5.4 Experimental Validation

5.4.1 Introduction

In this section, the observer-based neural network approach is applied to a four-tank system. The two types of NNs discussed above will be used. This case study will shed some light on the performance of the new approach.

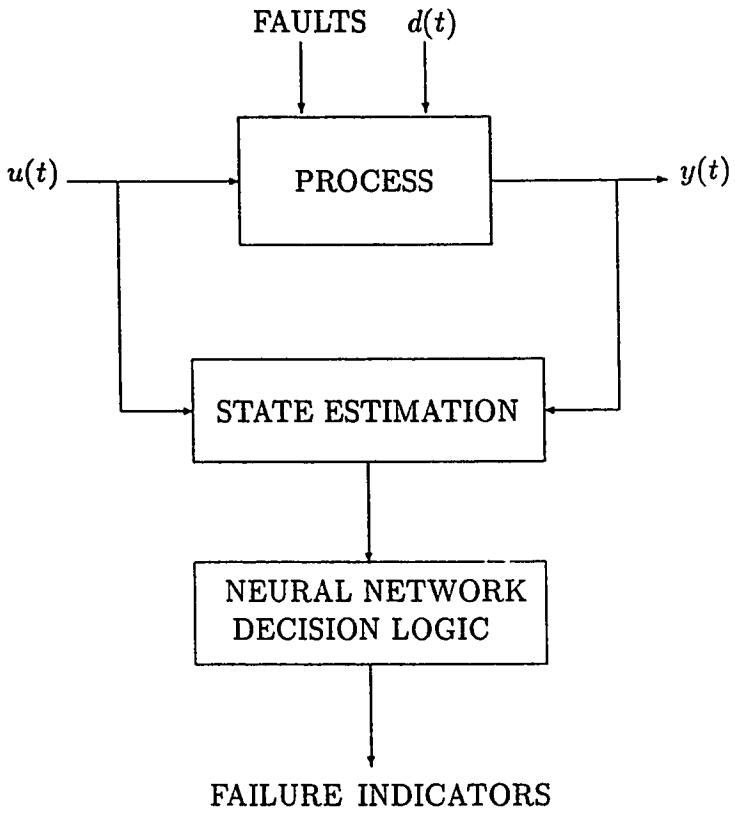


Figure 5.3: General structure of the new scheme

5.4.2 Description of the Experimental Setup

As stated above, the four-tank model is used to validate the new approach to detect and isolate leaks in the four-tanks. As shown in figure 5.3, the observer-based schemes are used to estimate the levels in each tank. The four robust observers, and the detection filters that were designed in chapter three, are used here to estimate the four-tank levels. The observer errors are generated and then processed by the NN (refer to Fig. 5.4). Three different NNs are used, together with the robust observers and the detection filters. Again, the objective of these three networks is to see which one of them performs better. The first type of neural nets is a three layer NN with four inputs $u_i(t)$, $i = 1, ..., 4$. The second type is also a three layer NN with eight inputs: $[u_i(t), u_i(t-1)]$, $i = 1, ..., 4$. The third type is a three layer NN with twelve inputs: $[u_i(t), u_i(t-1), u_i(t-2)]$, $i = 1, ..., 4$, (Fig. 5.1). The neural network inputs $u_i(t)$'s are defined as:

$$[u_i(t),..., u_i(t-d)] = \begin{cases} e_i(t),\ e_i(t-1),...,\ e_i(t-d), & for\ DF \\ \overline{e}_i(t),\ \overline{e}_i(t-1),...,\ \overline{e}_i(t-d), & for\ RBO \end{cases}, \quad i = 1\text{–}4 \quad (5.10)$$

where d is the delay time or width of the widow. The observer error signals e_i's and $\overline{e}$ are described as follows:

$$e_i(t) = [j_i^T z_i(t) + p_i^T y(t)] SR_i, \ for\ i = 1 - 4 \quad (5.11)$$

$$\overline{e}(t) = S[x(t) - \hat{x}(t)] \quad (5.12)$$

Where SR_i's ($SR_i = 5, i = 1, .., 4$) are scaling parameters used to scale the RBO errors between $[-1, 1]$. Similarly, S ($S = diag(-50, 50, 50, 50)$) is a scaling matrix

used to scale the DF error components between $[-1, 1]$.

Furthermore, the number of hidden nodes for the three NNs is nine processing nodes. This number was selected by trail and error. In the literature, there exist a mathematical rule (Kolmogorov theorem) [22] that states the number of processing nodes in the hidden layer is ($2\times$ # of input nodes $+1$) nodes. However, this rule does not give us the optimal number of hidden nodes for a NN. In other words, this rule does not tell us whether the resulting NN is the most efficient one to achieve our goals, or a smaller NN might perform the same task. Finally, the three NNs produce four alarm signals, each for one tank, that are used to monitor the four-tank system.

5.4.3 Learning and Validation Phases

Before training the three NNs, eight random failure events were generated. An example of such random failure events might be a series of leaks in tanks $1, 2, 4, 3, 1, 2$ respectively. It is very essential to make the trained network unbiased, which means that the NN is able to generalize all the sequences of failure events. Then, the system along with the robust observers and the detection filter were simulated for the same eight random events. The training patterns are generated by sampling $e_i(t)$ and $\overline{e}(t)$ signals (NN inputs) and then assigning the corresponding target values (NN outputs) for the NN outputs. For leak detection, an output node target value would be either 0 (no leak) or 1 (a particular leak). Then, the NNs are trained via the back-propagation algorithm.

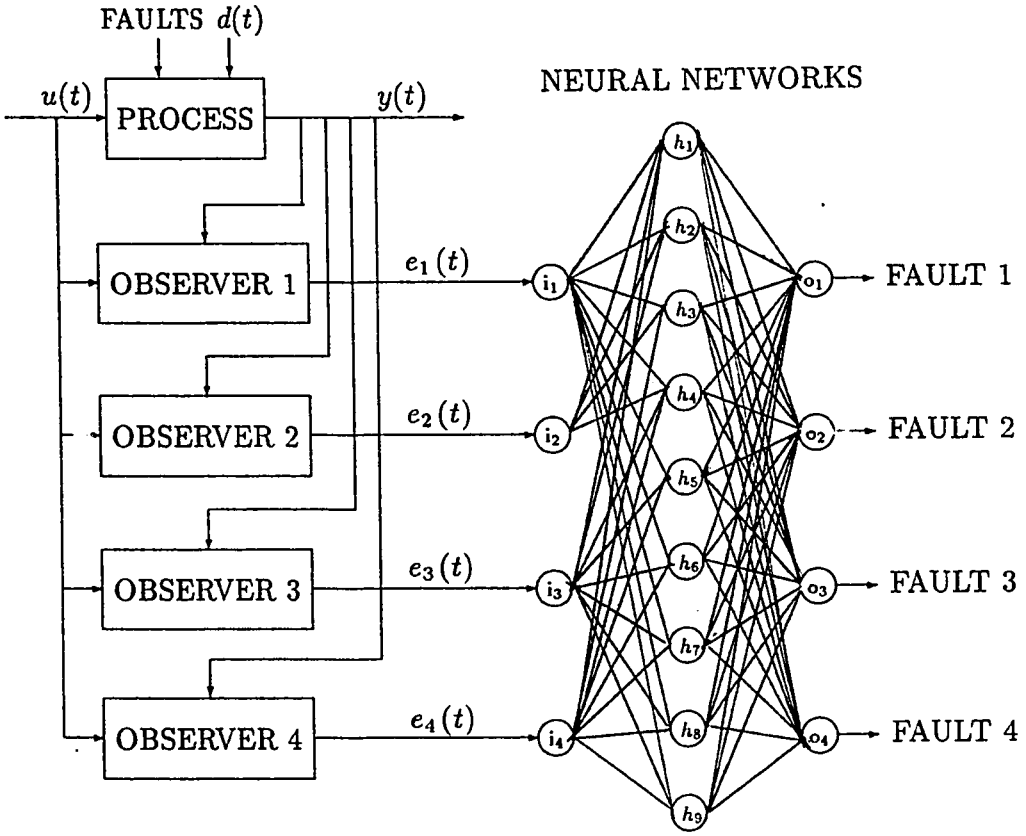


Figure 5.4: Fault detection via RBO/NNN approach

The initial weights of the NNs are selected randomly between $[-2, 2]$. A learning rate of 0.06 ($\zeta = 0.06$) and a momentum value of 0.2 ($\eta = 0.2$) are used in the training sessions [14]. The learning session is considered successful when the learning curve levels. Typical learning plots are shown in figures 5.5 and 5.6. These figures represent the training results for RBO/NN1 and DF/NN1. The number of iterations that were required taken to learn all training patterns for the six NNs is around 10000 iterations (around 36 hours on a 386 PC). The final weights of the NNs are tabulated in tables 5.1 and 5.2.

After the training phase, new input-output patterns are used to validate the trained networks. This step is very essential to make sure that the NN is able to identify new sequences of failure events. A typical validation test, that was undertaken, was to study the ability of the three NNs to detect leaks in the four-tank system. A typical simulation run that was conducted was to detect and isolate a leak in the base of tank 1. The system and the observers together with the NNs were simulated for a period of 100 *sec*. The leak occurs after 50 *sec* of the simulation time. The simulation run is started by a step input of $u = 1.0\ m^3 sec^{-1}$. A plot of the four alarm signals of the NN are shown in figures 5.7 and 5.8. As can be observed, the NN outputs of RBO/NN1 and DF/NN1 have detected and isolated the leak in tank 1 after 2 *sec*.

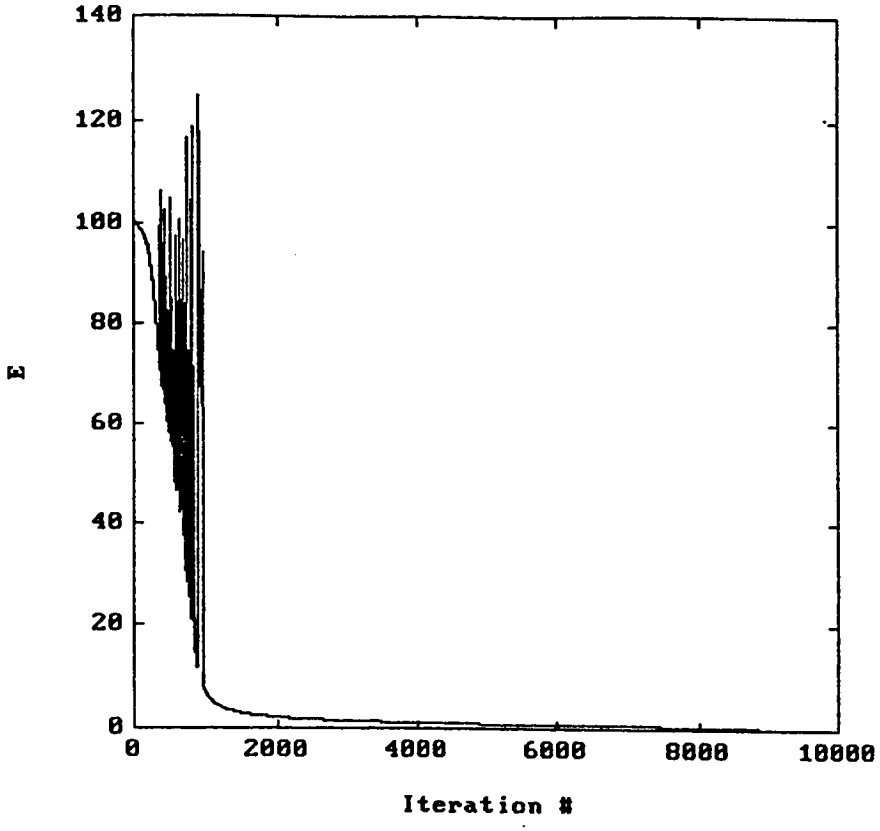


Figure 5.5: Learning curve for RBO/NN1

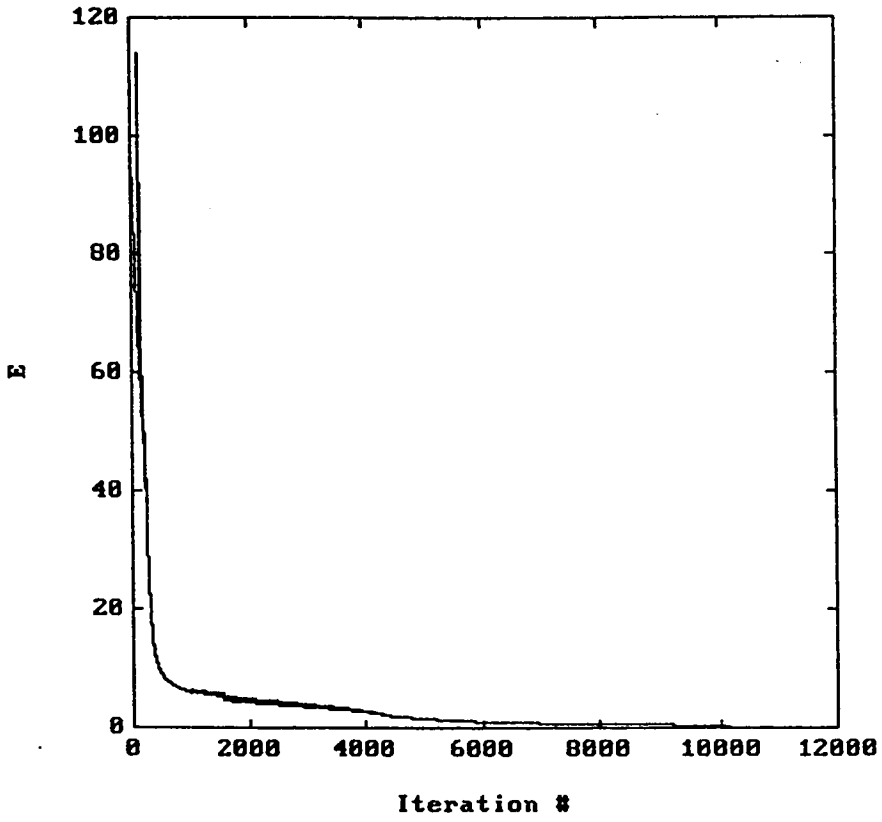


Figure 5.6: Learning curve for DF/NN1

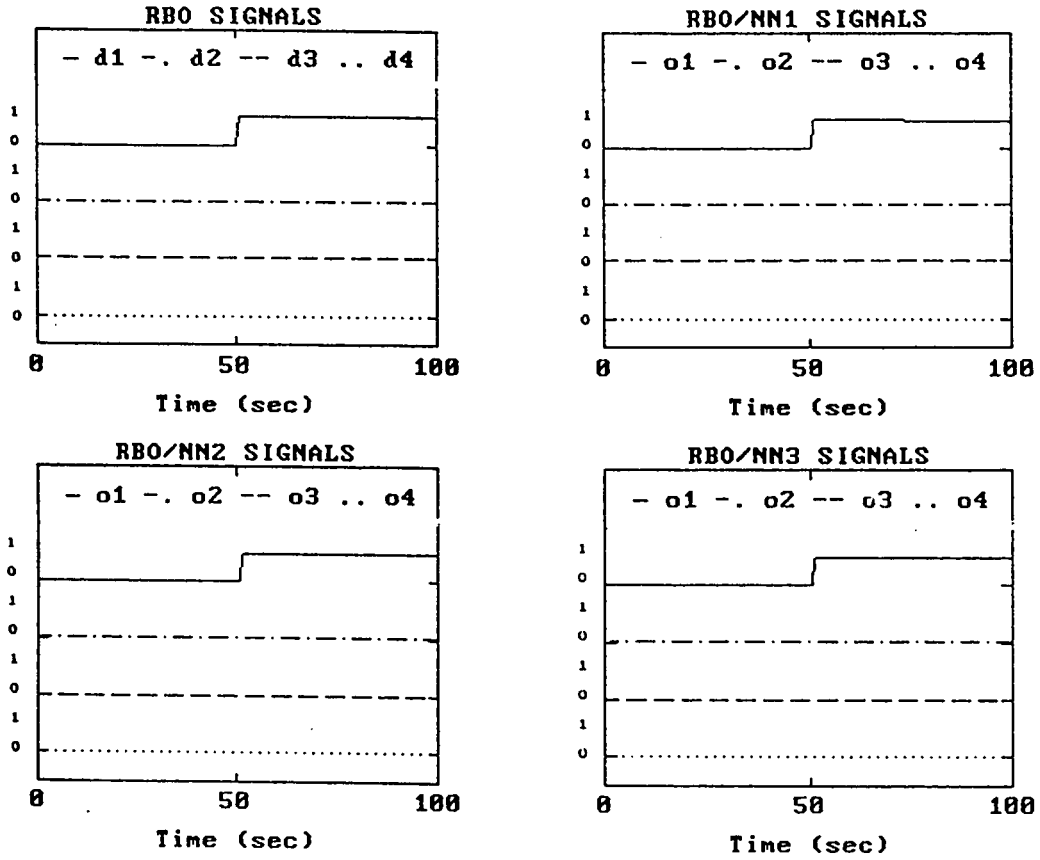


Figure 5.7: Response of RBO, RBO/NN1, RBO/NN2 & RBO/NN3 alarm signals for a leak in tank 1

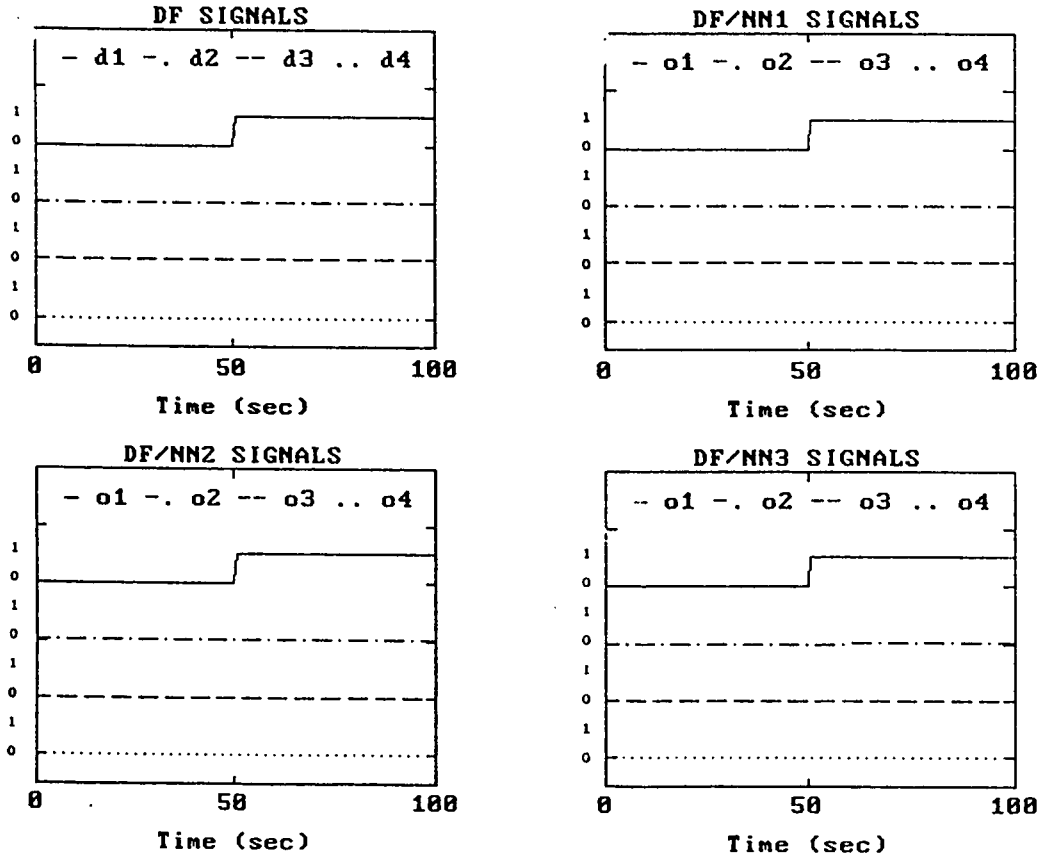


Figure 5.8: Response of DF, DF/NN1, DF/NN2 & DF/NN3 alarm signals for a leak in tank 1

Table 5.1: Weights for RBO/NN2 trained networks

Input Nodes				
Hidden Nodes	i_1	i_2	i_3	i_4
h_1	-2.27	-11.54	15.63	-1.55
h_2	8.54	-9.83	-16.20	18.28
h_3	-3.30	-0.58	0.47	0.60
h_4	-2.54	0.72	-1.41	-0.01
h_5	-17.63	8.34	2.86	8.84
h_6	-2.90	2.10	0.48	-2.26
h_7	10.51	9.06	1.71	-22.43
h_8	-14.61	6.45	1.56	8.40
h_9	-5.89	8.49	11.05	-14.72
Output Nodes				
Hidden Nodes	o_1	o_2	o_3	o_4
h_1	-1.59	11.74	-16.60	1.46
h_2	14.94	2.88	7.62	-21.51
h_3	-3.79	0.88	0.39	1.90
h_4	-2.95	-0.79	1.45	1.78
h_5	-18.62	-4.07	2.50	-3.67
h_6	-3.53	-1.15	1.24	4.74
h_7	12.94	-13.64	-5.39	14.22
h_8	-15.34	-2.69	2.92	-4.34
h_9	-6.25	-6.35	-7.68	14.32

Table 5.2: Weights for DF/NN2 trained networks

Input Nodes				
Hidden Nodes	i_1	i_2	i_3	i_4
h_1	-4.48	-13.37	18.92	-1.99
h_2	7.93	-8.44	-13.28	15.71
h_3	-5.14	-0.73	1.64	2.36
h_4	-4.28	1.77	-1.57	1.94
h_5	-17.30	3.98	2.46	11.79
h_6	-4.74	3.28	0.10	-0.90
h_7	11.41	10.47	-0.74	-21.98
h_8	-15.78	4.46	-0.46	12.54
h_9	-7.24	8.08	8.40	-10.98
Output Nodes				
Hidden Nodes	o_1	o_2	o_3	o_4
h_1	-8.79	18.48	-16.15	3.57
h_2	11.79	-1.12	6.02	-20.17
h_3	-8.94	1.37	-0.25	1.95
h_4	-7.71	-1.44	1.78	2.21
h_5	-18.90	-4.91	0.43	-4.29
h_6	-8.08	-1.32	1.24	5.52
h_7	20.06	-8.26	-8.32	11.27
h_8	-18.11	-5.20	3.65	-3.91
h_9	-9.29	-2.52	-7.69	14.50

5.4.4 Performance Under Perturbation

The observer-based fault detection approach in particular, the robust observers and the detection filter approaches has been shown in Chapter four, to be sensitive to parameter perturbation, which makes it unable to correctly detect and isolate faults, resulting in frequent false alarms. As an illustration, the diagnosis schemes of the four-tank system were analyzed under a perturbation δ in the input u, the inflow rate. The objective was to compute the range of δ via perturbation theory for which the robust observers and the detection filter will produce false alarms. The computed range of perturbation δ for the robust observers and the detection filter are $\{.1 < \delta < .09\}$ and $\{.02 < \delta < .03\}$ respectively. Thus, in face of parameter perturbations, the range of δ is quite small which will cause frequent false alarms in the diagnosis schemes. Nevertheless, it is important to point to the fact that the DF is much more sensitive to variations in δ then the RBO.

Similarly, the sensitivity of the observer-based neural network approach was investigated under perturbations δ in the inflow input u. The objective was to compare robustness between the observer-based approach and the observer/neural network based approach. To find the range of δ for the new scheme, the system was simulated for different values of δ. The range of δ for {RBO/NN1, RBO/NN2, RBO/NN3} and {DF/NN1, DF/NN2, DF/NN3} schemes are $\{-0.15 < \delta < M\}$ and $\{-0.07 < \delta < M\}$ respectively. The "M" is a large positive number that is used to prevent flooding in the four-tank system and to enable the NNs to be insensitive to flooding. In other words, the NNs are unable to detect flooding in tank i which might be caused by an increase in the inflow input to tank 1, or a

clog in the outlet pipe of tank i. However, the value of M will not be computed in this thesis. These results support the claim made earlier that the use of NN with the observer scheme increases robustness which in turn will reduce the rate of false alarms. However, the robustness gain achieved is quite limited!

In order to show a false alarm due to δ, the same simulation steps as in the validation phase were used. The response of the alarm signals for the different schemes are plotted in figures 5.9 and 5.10. In these figures, it is observed that while the observer scheme (DF and RBO schemes) gave a false alarm shortly after the perturbation appeared, the observer/neural network scheme did not produce any false alarm. Although the range of δ is small, one can increase it by training the NNs under δ perturbation.

For completeness, noise rejection, or noise robustness test, was carried for a uniformly distributed noise with intensities in $[-\alpha,\alpha]$. The aim was to find the maximum α that caused no false alarms. The results are summarized in table 5.3. These results suggest that the use of NN enhances the observer scheme in face of noise.

In addition to the previous two tests, the robustness of the new scheme can be improved further, if one uses the moving window neural network. By comparing the three types of NNs used in this thesis, it can be concluded that the third NN with two lagged inputs has shown relatively greater robustness than the other two NNs. For example, the range of δ in table 5.4 for the third NN with two lagged

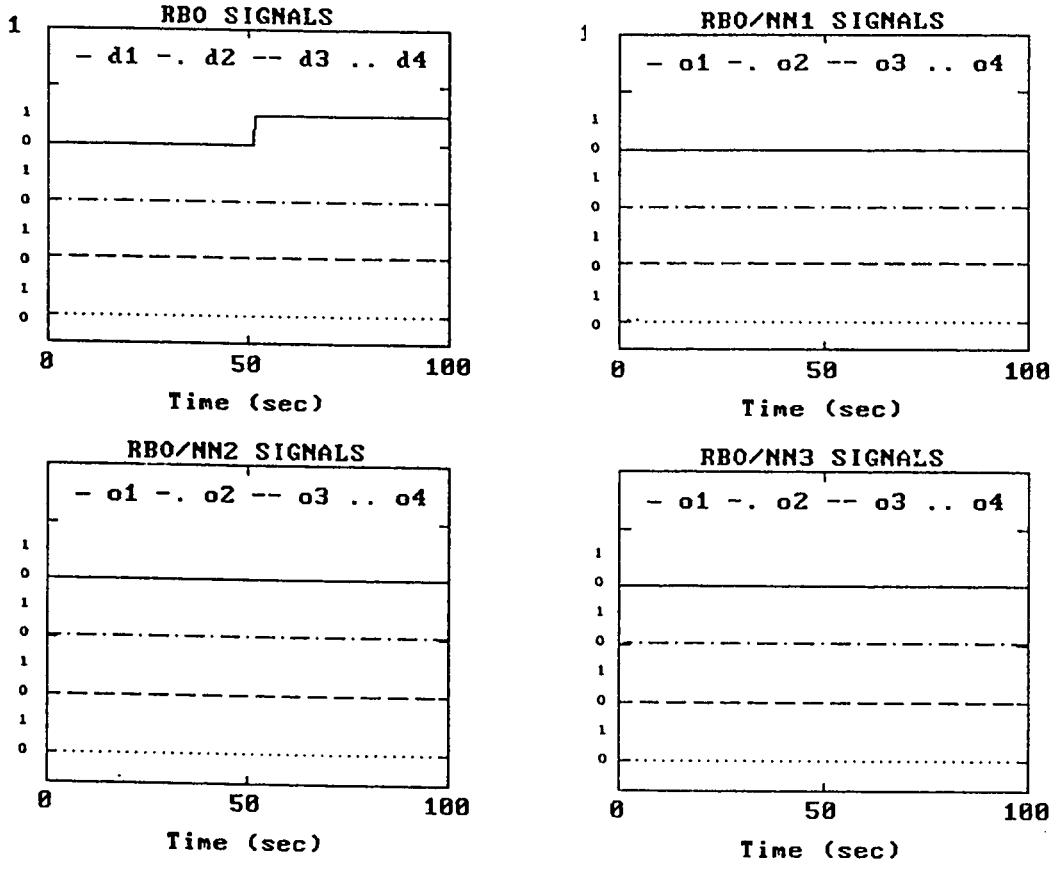


Figure 5.9: Response of RBO, RBO/NN1, RBO/NN2 & RBO/NN3 alarm signals for $\delta = 0.09$

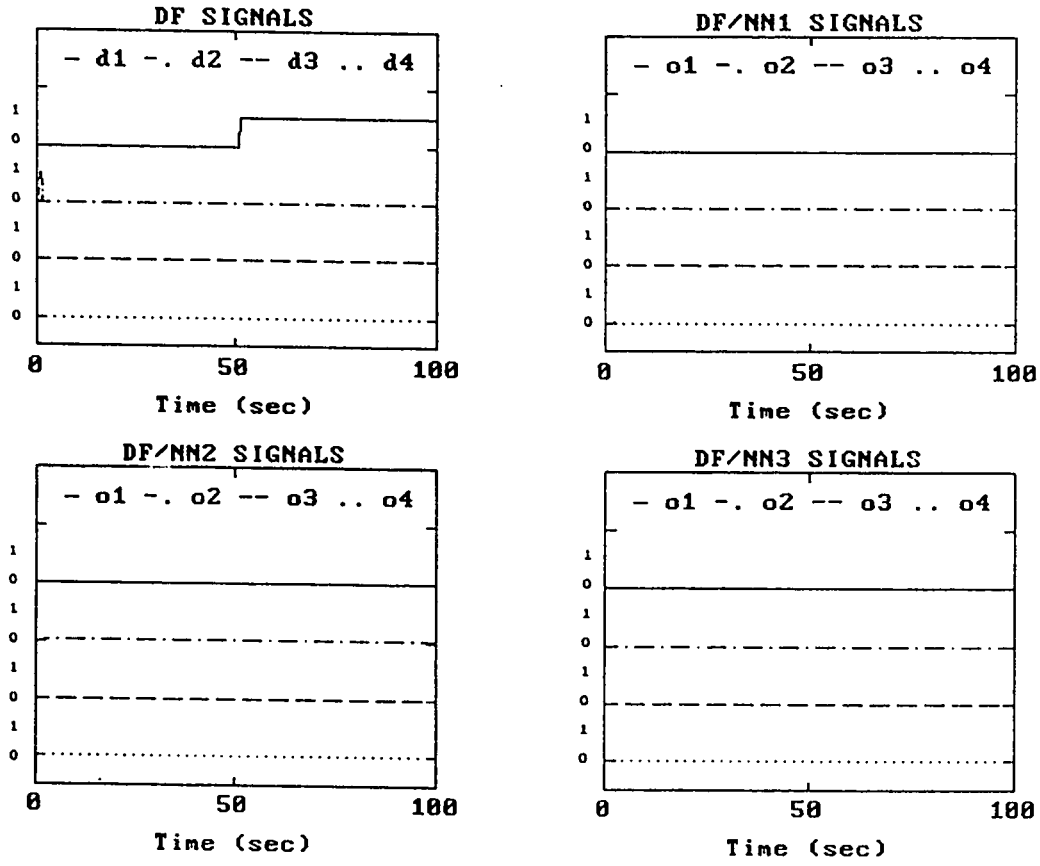


Figure 5.10: Response of DF, DF/NN1, DF/NN2 & DF/NN3 alarm signals for $\delta = 0.09$

inputs (i.e. NN3) is $\{-0.24 < \delta < M\}$, and for the other two NNs (i.e. NN1 and NN2) it is $\{-0.235 < \delta < M\}$. Finally, the robustness tests that were carried out have shown the feasibility to use NNs with the observer schemes.

A third test was conducted, to test the ability of the NNs to detect leaks under parameter perturbations. The system and NN schemes were simulated for various values of δ's, and simultaneously a leak in tank 1 was created. For negative values of δ's, all NN schemes were able to detect the leak in tank 1. This is because for negative values of δs (i.e., decrease in the inflow input to tank 1), or for a leak in tank 1, will decrease the level in tank 1. For positive values of δ, however, the RBO/NN and DF/NN schemes were able to detect the leak for $\delta < 0.15$, and for $\delta < 0.07$ respectively. The positive values of δ's correspond to flooding in tank 1, and under this condition, the NNs are insensitive to flooding, because they were not trained (designed) to do so.

Furthermore, additional tests must be investigated to identify the strengths and the limitations of the new scheme. Another test that may be conducted is to apply this new technique to another system, such as oil and gas transmission pipelines. The objective of this test would be to study the ability and performance of this new technique to detect failures in other dynamical systems.

5.5 Conclusions

A new failure detection scheme has been presented. The ability of this scheme to detect and isolate abrupt failures has been illustrated with the aid of a four-tank system. With this technique, more reliable and robust failure detection and isolation schemes can be achieved despite parameter perturbation and noise. This is due to the advantages of NNs, with their robustness to noise, parameter perturbations, system's non-linearities, and modeling errors, and generalization to other patterns.

In addition, two types of NNs were examined: naive and moving window neural networks. The use of moving window neural network provides extra robustness to parameter perturbations and noise. This is because the lagged inputs help to build an internal memory inside the NN which enables the NN to classify failure events based on past and current inputs. However, some of the main disadvantages of this new approach are extensive training time and structural complexity of the NN. These two drawbacks might be resolved by finding a smaller NN structure which in turn will decrease the robustness of the NN to noise and parameter perturbations.

The diagnosis scheme used to monitor the four-tank system was excellent in detecting and isolating leakages. However one disadvantage of the monitoring scheme, is that it is unable to detect flooding in the four-tank system, which is as important as detection of leakages in some dynamical systems. Consequently, for future work the NNs must be trained to detect both leakages and flooding in the tanks.

Finally, another point that was not examined in this work is the missed detection, which is inability of the FDI scheme to detect failures once they happened. This might be caused by two reasons: (i) small faults, (ii) incipient failures. Small faults are faults that are very difficult to detect. Incipient faults are slowly developing faults, which are mainly caused by corrosion.

Table 5.3: Robustness of the Observer/NNs-based schemes to inflow noise

scheme	uniform noise
RBO	$\alpha < 0.70$
DF	$\alpha < 0.65$
RBO/NN1	$\alpha < 1.31$
DF/NN1	$\alpha < 0.735$
RBO/NN2	$\alpha < 1.31$
DF/NN2	$\alpha < 0.735$
RBO/NN3	$\alpha < 1.31$
DF/NN3	$\alpha < 0.735$

Table 5.4: Robustness table for the Observer/NNs-based schemes

scheme	range of δ
RBO	$-0.10 < \delta < 0.09$
DF	$-0.03 < \delta < 0.02$
RBO/NN1	$-0.235 < \delta < M$
DF/NN1	$-0.10 < \delta < M$
RBO/NN2	$-0.235 < \delta < M$
DF/NN2	$-0.10 < \delta < M$
RBO/NN3	$-0.24 < \delta < M$
DF/NN3	$-0.10 < \delta < M$

Chapter 6

SUMMARY, CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE WORK

6.1 Introduction

The work described in this thesis involves the use of neural networks alongwith observer-based fault detection and isolation algorithms, to detect and isolate failures in dynamical systems. The ultimate goal of this scheme is to achieve the maximum robustness in the face of parameter perturbations, noise, and/or modeling errors. The basic steps of this scheme are generation of residual error signals through state estimator(s) and then processing these residuals through the neural network. This chapter presents the summary and conclusions of this research and provides recommendations for future work.

6.2 Summary

Two fault detection and isolation techniques have been studied in this thesis: the robust observer, and the detection filter. It was shown in Chapter four that these two schemes are highly sensitive to parameter perturbation, resulting in frequent false alarms. Therefore, in the event of a failure alarm, there is no way to find out whether the alarm is due to a fault or to a parameter perturbation in the process. Using standard perturbation techniques, it was shown that the design parameters of the robust observer, and the detection filter, have great influence on the sensitivity of these two schemes.

Furthermore, it was shown that, by assigning the eigenvalues of the observer-based fault detection schemes far to the left of the system's eigenvalues, the sensitivity of the observers will increase. Consequently, any small parameter perturbation, noise, or modeling errors, will be magnified, which in turn will, most likely, cause false alarms. This was illustrated by the four-tank model.

Because of the main advantages offered by neural networks such as generalization, robustness to noise, and ability to learn a specific non-linear pattern, neural networks have shown promise for applications to different areas of engineering [14]. In the area of failure detection [2]; [13], for example, the neural network is trained to learn a particular set of failure patterns in the system, so that when a failure appears, then the neural network will produce an alarm signal. Additional references on recent applications of neural networks in the area of failure detection are tabulated in the references starting from [23]. Although this approach is good

for fast dynamical systems, such as chemical reactors with fast chemical reactions, it has some problems with slow dynamical systems such as distillation columns. Some of these problems are:

1. In order to cover the pertinent system's dynamics, by sampling the system outputs, very long training time is needed.
2. The fault detection time is large, due to the slow response of the systems to such events.

In order to alleviate these problems, a novel approach has been proposed in Chapter five. Basically, this solution proposes the replacement of the detection logic with artificial neural networks, while keeping the remaining components of the observer-based fault detection schemes unchanged. As it has been demonstrated with the aid of the four-tank system, neural networks can not only decrease the sensitivity of the observer-based fault detection schemes to parameter perturbation, but they can also provide robustness against noise and modeling errors. The residuals of the observer-based scheme are processed through the neural network in a dynamic fashion. This means that as soon as the residuals are sampled during a simulation run for times $(t, t+1, ...)$, they are processed through the neural network.

In order to improve the performance of the neural network, three types of neural networks were used. The first neural network is called naive neural network

(NNN) which uses residual samples at time t. The second and third neural networks are called moving window neural networks (MWNN) which employ residual samples at times $(t, t-1)$ and $(t, t-1, t-2)$ respectively. As was expected, the performance of the MWNN's is much better than the NNN. This is because, the decision logic in the case of the NNN is based on the current values of the residuals only. On the other hand, the decision logic in the MWNN is based on the current and the past values of the residuals.

6.3 Conclusions

The sensitivity analysis of the observer-based fault detection scheme has shed some light on how to design observers that are less sensitive to parameter perturbations while sensitive to faults in dynamical systems. The observer-based neural network approach enables engineers to use simple observers, such as robust observers or detection filters, over the special type of observers, which is computationally very complex.

In addition, it has been shown in Chapter five that the observer-based neural network approach can detect and isolate failures in the four-tank system as well as the observer-based approach and even better. Hence, one would expect that the new scheme will have promising applications in failure detection and isolation in complex dynamical systems such as distillation columns, chemical reactors, navigation systems, transmission pipelines, etc.

The robustness of the new fault detection scheme, to parameter perturbation, has been demonstrated on the four-tank system. With this technique, the range of parameter perturbation has been increased from $\delta = .09$ to $\delta = .25$ for robust observers, and from $\delta = .02$ to $\delta = .1$ for detection filter, which implies that the rate of false alarms has been decreased. Moreover, a uniform noise between $[-\gamma, \gamma]$ has been added to the first measurement sensor in tank 1 with amplitude $\pm\ \gamma$. As γ was increased to 0.025 the new diagnosis system was robust to this white noise. However, if γ is increased beyond 0.03, the new diagnosis scheme declares false alarm.

Furthermore, the neural networks used in the four-tank model offer a few improvements over other neural network applications, such as in pattern recognition and parallel processing. These are as follows:

1. The training time of the neural networks requires less than 15,000 iterations.
2. The number of training patterns used are less than 250 patterns.
3. The size of the used neural networks is relatively small.
4. The failure alarm is reported promptly ($\approx$ 2.0 *sec*). If neural networks only are used as a failure detection and isolation scheme with the four-tank system, the failure alarm will be reported after about 350 to 400 seconds

On the other hand, the observer-based neural network approach has some disadvantages. One of these is the training time which is needed by the back-

propagation algorithm. In order to decrease the training time, the number of hidden nodes had to be made as few as possible. This was accomplished by trial and error. First, a reduced number of hidden nodes was selected, then the neural network was trained. If the neural network learns the training patterns and recognizes the validation set, then the minimum number of hidden nodes is reached. However, if the neural network doesn't learn the training patterns, then the number of hidden nodes is increased by one and the previous steps are repeated.

6.4 Recommendations for Future Work

In the course of this study, a number of areas for further research have been identified. Those which appear to offer the greatest potential yield are listed below:

- Investigate the application of this new scheme to other dynamical systems such as detection and localization of leaks in oil and gas transmission pipelines in Saudi Arabia. This is an important application because of safety, environment and economy.

- It will be of interest to study the applicability of this new scheme to detect incipient faults (slowly developing faults) in dynamical systems. These faults are of major relevance in connection with maintenance problems where early detection of worn equipment is needed. In this case, the faults are typically small and not as easy to detect, but the detection time is of minor importance in maintenance applications and may, therefore, be large.

- Another area that may be studied is to find which neural network approaches are easier to apply than the back-propagation algorithm used in this thesis.

Appendix A

THE FOUR-TANK NON-LINEAR SYSTEM

To test the fault detection schemes, a four-tank non-linear system, shown in figure A.1, will be used. A similar model has been used by Ge and Fang [8] for leak failure detection. In order to develop a model for this system, a mass balance is performed around each tank as follows :

$$Water\ accumulated\ in\ tank\ i\ =\ input\ -\ output \tag{A.1}$$

For tank i the mass balance equation will be

$$a_1\dot{x}_1(t) = (f_{in})_i(t) - (f_{out})_i(t) \tag{A.2}$$

where $(f_{in})_1 = u$ is the input to the first tank, and for the other tanks they are as follows

$$\begin{aligned} (f_{in})_2 &= s_1\sqrt{2p_1} = s_1\sqrt{2\rho g(x_1 - x_2)} \\ (f_{in})_3 &= s_2\sqrt{2p_2} = s_2\sqrt{2\rho g(x_2 - x_3)} \end{aligned}$$

$$(f_{in})_4 = s_3\sqrt{2p_3} = s_1\sqrt{2\rho g(x_3 - x_4)}$$

The outputs from tanks 1, 2, 3, and 4 are as follows

$$\begin{aligned}
(f_{out})_1 &= s_1\sqrt{2p_1} = s_1\sqrt{2\rho g(x_1 - x_2)} \\
(f_{out})_2 &= s_2\sqrt{2p_2} = s_2\sqrt{2\rho g(x_2 - x_3)} \\
(f_{out})_3 &= s_3\sqrt{2p_3} = s_3\sqrt{2\rho g(x_3 - x_4)} \\
(f_{out})_4 &= s_4\sqrt{2p_4} = s_4\sqrt{2\rho g x_4}
\end{aligned}$$

Where

$$\begin{aligned}
(f_{in})_i &= \textit{input to tank } i \\
(f_{out})_i &= \textit{output from tank } i \\
p_i &= \textit{pressure drop at the outlet of tank } i \\
\rho &= \textit{density of water } (1.0\ Kg/m^3) \\
g &= \textit{gravitational acceleration constant } (9.81\ m/sec^2) \\
s_i &= \textit{cross} - \textit{sectional area of outlet pipe of tank } i \\
a_i &= \textit{cross} - \textit{sectional area of tank } i \\
u &= \textit{a scalar input}
\end{aligned}$$

Let the non-linear model be written as follows

$$\dot{x}_1(t) = -b_1\sqrt{x_1(t) - x_2(t)} + \frac{1}{a}u(t) \tag{A.3}$$

$$\dot{x}_2(t) = b_1\sqrt{x_1(t) - x_2(t)} - b_2\sqrt{x_2(t) - x_3(t)} \tag{A.4}$$

$$\dot{x}_3(t) = b_2\sqrt{x_2(t) - x_3(t)} - b_3\sqrt{x_{3(t)} - x_4(t)} \tag{A.5}$$

$$\dot{x}_4(t) = b_3\sqrt{x_3(t) - x_4(t)} - b_4\sqrt{x_4(t)} \tag{A.6}$$

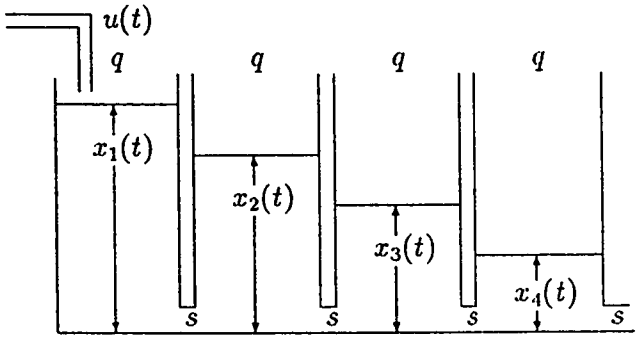


Figure A.1: The non-linear four-tank system

where $b_i = \frac{s_i}{a}\sqrt{2\rho g}$ and $a = a_i$, for $i = 1, 2, 3, 4$

The nominal point x_n around which the model will be linearized is

$$x_n = [x_{n1}, x_{n2}, x_{n3}, x_{n4}]^T = [4, 3, 2, 1]^T . \left[\frac{u_n}{b_i a}\right]^2 \tag{A.7}$$

where a, the nominal input u_n, and b_i have the following values

$$\begin{aligned} a &= 5\ m^2 \\ u_n &= 1\ m\,sec^{-1} \\ s_i &= \frac{1}{\sqrt{2 \times 1 \times 9.81}} = 0.2258\ m^2 \\ b_i &= \frac{s_i}{a}\sqrt{2 \times 1 \times g} = \frac{\sqrt{2 \times 1 \times 9.81}}{\sqrt{2 \times 1 \times 9.81} \times 5} = 0.2\ m^{1/2} sec^{-1} \end{aligned}$$

Thus, x_n is $(4, 3, 2, 1)^T$.

Let the non-linear model be represented by

$$f(x(t), u(t)) = \begin{bmatrix} -b_1\sqrt{x_1(t) - x_2(t)} + \frac{u(t)}{a} \\ b_1\sqrt{x_1(t) - x_2(t)} - b_2\sqrt{x_2(t) - x_3(t)} \\ b_2\sqrt{x_2(t) - x_3(t)} - b_3\sqrt{x_3(t) - x_4(t)} \\ b_3\sqrt{x_3(t) - x_4(t)} - b_4\sqrt{x_4(t)} \end{bmatrix} \tag{A.8}$$

Now taking the partial derivatives of (A.8) and evaluating them at x_n yields

$$\frac{\partial f}{\partial x} = \begin{bmatrix} -\frac{b_1\alpha_1}{2} & \frac{b_1\alpha_1}{2} & 0 & 0 \\ \frac{b_1\alpha_1}{2} & \beta_1 & \frac{b_2\alpha_2}{2} & 0 \\ 0 & \frac{b_2\alpha_2}{2} & \beta_2 & \frac{b_3\alpha_3}{2} \\ 0 & 0 & \frac{b_3\alpha_3}{2} & \beta_3 \end{bmatrix} \tag{A.9}$$

$$\frac{\partial f}{\partial u} = \begin{bmatrix} \frac{1}{a} \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{A.10}$$

where

$$\begin{aligned}
\alpha_1 &= \frac{1}{\sqrt{x_1 - x_2}} = 1.0\ m^{-1/2} \\
\alpha_2 &= \frac{1}{\sqrt{x_2 - x_3}} = 1.0\ m^{-1/2} \\
\alpha_3 &= \frac{1}{\sqrt{x_3 - x_4}} = 1.0\ m^{-1/2} \\
\alpha_4 &= \frac{1}{\sqrt{x_4}} = 1.0\ m^{-1/2} \\
\beta_1 &= -\frac{b_1\alpha_1 + b_2\alpha_2}{2} = \frac{1}{5} \\
\beta_2 &= -\frac{b_2\alpha_2 + b_3\alpha_3}{2} = \frac{1}{5} \\
\beta_3 &= -\frac{b_3\alpha_3 + b_4\alpha_4}{2} = \frac{1}{5}
\end{aligned}$$

Substituting α_i's and b_i's in the linearized model, yields

$$\delta\dot{x}(t) = \frac{1}{10}\begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \end{bmatrix}\delta x(t) + \begin{bmatrix} 0.2 \\ 0 \\ 0 \\ 0 \end{bmatrix}\delta u(t) \tag{A.11}$$

where $\delta x(t) = [\delta x_1(t)\ \delta x_2(t)\ \delta x_3(t)\ \delta x_4(t)]^T$.

Then, the vector $x(t)$ is found as follows

$$x(t) = x_n + \delta x(t) \tag{A.12}$$

$$x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \\ 2 \\ 1 \end{bmatrix} + \begin{bmatrix} \delta x_1(t) \\ \delta x_2(t) \\ \delta x_3(t) \\ \delta x_4(t) \end{bmatrix} \tag{A.13}$$

Now if the non-linear system is subjected to failures such as leakages, or cloggings, then modeling of these failures is important to account for them. Hence, a leak in tank 1, for example, is modeled by doing a mass balance around it, which leads to

$$\dot{x}_1(t) = \frac{u(t)}{a} - b_1\sqrt{x_1(t) - x_2(t)} - (f_{Leak})_1 \tag{A.14}$$

where

$$\begin{aligned} (f_{Leak})_1 &= \overline{a}\sqrt{2\, pressure\ at\ the\ base\ of\ tank\ 1} = \overline{a}\sqrt{2gx_1(t)} \\ \overline{a} &= cross-sectional\ area\ of\ the\ leak\ in\ tank\ 1 \end{aligned}$$

Therefore, leaks in the other tanks can be modelled in the same way, to be

$$(F_{Leak})_i = \overline{a}_i\frac{\sqrt{2gx_i(t)}}{a},\ for\, i = 1,2,3,4 \tag{A.15}$$

Similarly, in order to model a clog at the outlet pipe of say, tank 1, a mass balance around tank 1 and tank 2 must be done. Thus, the resulting equations are

$$\dot{x}_1(t) = -b_1^*\sqrt{x_1(t) - x_2(t)} + \frac{u}{a} \tag{A.16}$$

$$\dot{x}_2(t) = b_1^*\sqrt{x_1(t) - x_2(t)} - b_2\sqrt{x_2(t) - x_3(t)} \tag{A.17}$$

where

$$b_1^* = [s_1 - s_1^*]\frac{\sqrt{2g}}{a} = b_1 - s_1\frac{\sqrt{2g}}{a}$$

$$s_1^* = \textit{reduction in cross} - \textit{sectional area of tank 1 outlet}$$

Or re-writing equations (A.16) and (A.17) in another form

$$\dot{x}_1(t) = -b_1\sqrt{x_1(t) - x_2(t)} + \frac{u}{a} + s_1^*\frac{\sqrt{2g[x_1(t) - x_2(t)]}}{a}$$

$$\dot{x}_2(t) = b_1\sqrt{x_1(t) - x_2(t)} - b_2\sqrt{x_2(t) - x_3(t)} - s_1^*\frac{\sqrt{2g[x_1(t) - x_2(t)]}}{a}$$

Unlike leakage failure, a clog failure at the outlet of tank 1 effects both states x_1 and x_2 in the model. Now various types of faults can be represented in the following way.

$$\dot{x}(t) = f(x(t), u(t)) + \sum_{i=1}^{4} L_i m_i(t) + \sum_{i=1}^{4} L_{i+4} m_{i+4}(t) \qquad \text{(A.18)}$$

where $L_i m_i(t)$ and $L_{i+4} m_{i+4}(t)$ are the terms accounting for leaks and clogs in tanks 1, 2, 3, and 4. L_i and L_{i+4} are vector matrices (failure signature matrices) listed in table A.1. $m_i(t)$ and $m_{i+4}(t)$ are failure functions defined in table A.2 (when failure i does not occur, then $m_i(t) = 0$).

Table A.1: Failure signature matrices

i	1	2	3	4	5	6	7	8
1	1	0	0	0	1	0	0	0
2	0	1	0	0	-1	1	0	0
3	0	0	1	0	0	-1	1	0
4	0	0	0	1	0	0	-1	1

Table A.2: Leakages and cloggings failure functions

$m_i(t)$	=	$\frac{-\overline{a}_i\sqrt{2gx_i(t)}}{a}$, $for\ i = 1,2,3,4$
$m_{i+4}(t)$	=	$\frac{-s_i^*\sqrt{2g[x_i(t)-x_{i+1}(t)]}}{a}$, $for\ i = 1,2,3$
$m_8(t)$	=	$\frac{-s_4^*\sqrt{2gx_4(t)}}{a}$
Where		
$\overline{a}_i$	=	cross-sectional area of leakage in tank i
s_i^*	=	reduction in cross-sectional area of outlet tank i
a	=	5 m^2

Appendix B

SOFTWARE PROGRAMS

This Appendix briefly describes nine Matlab programs. In each program listing, there is a short introduction and explanation of the input/output parameters for the program. Each program uses several subroutines. Table B.1 shows the relationship between each program and these subroutines. These programs may be divided into two categories: failure detection and isolation programs, and a neural network simulator program.

I. The failure detection and isolation programs

These are eight programs, used to detect and isolate failures in the four-tank non-linear system, via the observer and the observer-based neural network approaches. These programs are limited to the following:

1. The dimension of the system used is four.
2. The number of robust observers used is four.
3. One detection filter is used.

4. The neural network used is a three layer neural net.

In order to use these programs to another system, one has on modify these programs as follows:

1. After modeling the system, input the model differential equations into files FE1 and FE2.
2. Then, design a number of observers that are used as state estimators. The objective of this is to monitor any abnormality in the system.
3. Train the neural networks using EXS71 program. At the end of the training, store the final weights of the networks.

II. The neural network simulator program

This is the EXS71 program which is used as a three layer neural network simulator via back-propagation algorithm [14].

Table B.1: Subroutines used by the various programs

Program Name	Subroutine Names				
	ODD45	PROP	FPROPG	FE1	FE2
ERO1	x			x	
ESF1	x				x
ENRO1	x	x		x	
ENSF1	x	x			x
ENRO3	x	x		x	
ENSF3	x	x			x
ENRO5	x	x		x	
ENSF5	x	x			x
EXS71		x	x		

PROGRAMS LISTING

```
%
%                        *** ERO1 ***
%
%  This programe is used to simulate RBO (robust observers)
%  fault detection scheme (Fast Simulation).
%
%
%
clc
j2=input('Enter # of failure modes = ');
tw=input('{ti1;ti2;...;tij;tf} = ');
ti=tw(1:j2,:);
tf=tw(2:j2+1,:);
xm=tw(j2+1,:);
r1=5*rand(j2,1)
xo=[2.5;1.5;1;.5;14.4275;-9.12195;-6.0813;-3.045;0];
clg
% Input data
p1=[-5.771 0 0 0];
p2=[0 6.0813 0 0];
p3=[0 0 6.0813 0];
p4=[0 0 0 6.0907];
ee=[];tt=[];xx=[];f11=[];f22=[];f33=[];f44=[];
%
for j1=1:j2
%
 if r1(j1,1)>=0 & r1(j1,1)<1;a=0;c=0;d=0;e=0;end
 if r1(j1,1)>=1 & r1(j1,1)<2;a=-.015;c=0;d=0;e=0;
 disp('  *** WARNNING ***');
 disp(' ');disp('  TANK 1 IS LEAKING');disp('');end
 if r1(j1,1)>=2 & r1(j1,1)<3;a=0;c=-.02;d=0;e=0;
 disp('  *** WARNNING ***');
 disp(' ');disp('  TANK 2 IS LEAKING');disp('');end
 if r1(j1,1)>=3 & r1(j1,1)<4;a=0;c=0;d=-.02;e=0;
 disp('  *** WARNNING ***');
```

```
disp(' ');disp('  TANK 3 IS LEAKING');disp('');end
if r1(j1,1)>=4 & r1(j1,1)<5;a=0;c=0;d=0;e=-.02;
disp('  *** WARNING ***');
disp(' ');disp('  TANK 4 IS LEAKING');disp('');end
%
tii=ti(j1,1);tff=tf(j1,1);
[t1,x1]=odd45('fe1',tii,tff,xo,1.0e-6,0,a,c,d,e);
[m1,n1]=size(x1);xo=x1(m1:m1,:);xo=xo';
x1=x1(:,1:8);
e1=5*[[p1;p2;p3;p4] eye(4)]*x1';
ee=[ee;e1'];tt=[tt;t1];xx=[xx;x1];
if a==-.015;f1=1;else f1=0;end;
if c==-.02;f2=1;else f2=0;end;if d==-.02;f3=1;else f3=0;end;
if e==-.02;f4=1;else f4=0;end;
f11=[f11;f1*ones(m1,1)];
f22=[f22;f2*ones(m1,1)];
f33=[f33;f3*ones(m1,1)];
f44=[f44;f4*ones(m1,1)];
end
e=ee;t=tt;
m1=length(e);
d11=[];d22=[];d33=[];d44=[];
%
x=xx(:,1:4);
x1=x(:,1:1);x2=x(:,2:2);x3=x(:,3:3);x4=x(:,4:4);
subplot(221)
axis([0 t(m1) 0 5])
plot(t,x1,'-',t,x2,'-.',t,x3,'--',t,x4,':'),hold
plot(t,x1,'-'),xlabel('TIME'),title('SYSTEM STATES ')
hold
%
l1=length(f11);l2=ones(l1,1);
axis([0 t(l1) -1 8])
f1=f11+6*l2;f2=f22+4*l2;f3=f33+2*l2;f4=f44;
plot(t,f1,'-',t,f2,'-.',t,f3,'--',t,f4,':'),hold
plot(t,f1,'-'),xlabel('TIME')
title(' FAILURE FUNCTIONS')
```

```
hold;
e1=e(:,1:1);e2=e(:,2:2);e3=e(:,3:3);e4=e(:,4:4);
axis([0 t(m1) -1 1])
plot(t,e1,'-',t,e2,'-.',t,e3,'--',t,e4,':'),hold
plot(t,e1,'-'),xlabel('TIME'),title(' ERRORS');hold
%
for i=1:m1
if abs(e1(i))>=.125;d1=1;end;if abs(e1(i))<.125;d1=0;end
if abs(e2(i))>=.125;d2=1;end;if abs(e2(i))<.125;d2=0;end
if abs(e3(i))>=.125;d3=1;end;if abs(e3(i))<.125;d3=0;end
if abs(e4(i))>=.125;d4=1;end;if abs(e4(i))<.125;d4=0;end
d11=[d11;d1];d22=[d22;d2];d33=[d33;d3];d44=[d44;d4];
end
[m1,n1]=size(d1);dd=ones(m1,1);
d1=d11+6*dd;d2=d22+4*dd;d3=d33+2*dd;d4=d44;
l1=length(t);
axis([0 t(l1) -1 8])
plot(t,d1,'-',t,d2,'-.',t,d3,'--',t,d4,':'),hold
plot(t,d1,'-'),xlabel('TIME'),title('ALARM SIGNALS')
pause
hold
% ****************************************************
%                  *** ESF1 ***
%
%  This program is used to simulate DF (detection filter)
%  fault detection scheme (Fast simulation).
%
clc
j2=input('Enter # of failure modes (j) =  ');
tw=input('{ti1;ti2;...;tij;tf} = ');
ti=tw(1:j2,:);
tf=tw(2:j2+1,:);
xm=tw(j2+1,:);
clg
r1=5*rand(j2,1)
xo=[2.5;1.5;1;.5;2.5;1.5;1.;.5;0];
% Input data
```

```
ee=[];tt=[];xx=[];f11=[];f22=[];f33=[];f44=[];
%
for j1=1:j2
%
if r1(j1,1)>=0 & r1(j1,1)<1;a=0;c=0;d=0;e=0;end
if r1(j1,1)>=1 & r1(j1,1)<2;a=-.015;c=0;d=0;e=0;
disp('  *** WARNING ***');
disp(' ');disp('  TANK 1 IS LEAKING');disp('');end
if r1(j1,1)>=2 & r1(j1,1)<3;a=0;c=-.02;d=0;e=0;
disp('  *** WARNING ***');
disp(' ');disp('  TANK 2 IS LEAKING');disp('');end
if r1(j1,1)>=3 & r1(j1,1)<4;a=0;c=0;d=-.02;e=0;
disp('  *** WARNING ***');
disp(' ');disp('  TANK 3 IS LEAKING');disp('');end
if r1(j1,1)>=4 & r1(j1,1)<5;a=0;c=0;d=0;e=-.02;
disp('  *** WARNING ***');
disp(' ');disp('  TANK 4 IS LEAKING');disp('');end
%
tii=ti(j1,1);tff=tf(j1,1);
[t1,x1]=odd45('fe2',tii,tff,xo,1.0e-6,0,a,c,d,e);
[m1,n1]=size(x1);xo=x1(m1:m1,:);xo=xo';
x1=x1(:,1:8);
x=x1(:,1:4);z=x1(:,5:8);e1=x-z;
ee=[ee;e1];tt=[tt;t1];xx=[xx;x1];
f1=0;f2=0;f3=0;f4=0;
if a==-.015;f1=1;end
if c==-.02;f2=1;end
if d==-.02;f3=1;end
if e==-.02;f4=1;end
ff1=f1+6;ff2=f2+4;ff3=f3+2;
f11=[f11;ff1*ones(m1,1)];
f22=[f22;ff2*ones(m1,1)];
f33=[f33;ff3*ones(m1,1)];
f44=[f44;f4*ones(m1,1)];
end
t=tt;e=ee;
subplot(221)
```

```
x1=xx(:,1);x2=xx(:,2);x3=xx(:,3);x4=xx(:,4);
axis([0 xm 0 5])
plot(t,x1,'-',t,x2,'-.',t,x3,'--',t,x4,':'),hold
plot(t,x1,'-'),xlabel('TIME'),title('SYSTEM STATES')
hold
%
axis([0 xm -1 8])
plot(t,f11,'-',t,f22,'-.',t,f33,'--',t,f44,':')
hold
plot(t,f11,'-'),xlabel('TIME')
title('FAILURE FUNCTIONS'),hold
e1=-50*e(:,1);e2=50*e(:,2);e3=50*e(:,3);e4=50*e(:,4);
axis([0 xm -1 1])
plot(t,e1,'-',t,e2,'-.',t,e3,'--',t,e4,':'),hold
plot(t,e1,'-'),xlabel('TIME'),title('OBSERVER ERRORS')
hold
m=length(ee);d11=[];d22=[];d33=[];d44=[];sth=180/pi;
for i=1:m
sq=(e1(i)^2+e2(i)^2+e3(i)^2+e4(i)^2+.001)^.5;
b11=acos(abs(e1(i))/sq)*sth;
b22=acos(abs(e2(i))/sq)*sth;
b33=acos(abs(e3(i))/sq)*sth;
b44=acos(abs(e4(i))/sq)*sth;
d1=0;d2=0;d3=0;d4=0;
if b11<=40;d1=1;end
if b22<=15.0;d2=1;end
if b33<=40;d3=1;end
if b44<=40;d4=1;end
%
d5=d1+6;d6=d2+4;d7=d3+2;
d11=[d11;d5];d22=[d22;d6];d33=[d33;d7];d44=[d44;d4];
end
%
d1=d11;d2=d22;d3=d33;d4=d44;
axis([0 xm -1 8])
plot(t,d1,'-',t,d2,'-.',t,d3,'--',t,d4,':'),hold
plot(t,d1,'-'),xlabel('TIME')
```

```
title('DETECTION LOGIC'),pause
hold
% ****************************************************
%                  *** ERR01 ***
%
%  This programe is used to simulate RBO/WW1 fault
%  detection scheme (Fast simulation).
%
%j2=input('Enter # of Failure modes = ');
tw=input('{ti1;ti2;...;tij;tf} = ');
ti=tw(1:j2,:);
tf=tw(2:j2+1,:);
r1=5*rand(j2,1)
xm=tw(j2+1,:);
xo=[2.5;1.5;1;.5;14.4275;-9.12195;-6.0813;-3.045;0];
clg
% Input data
p1=[-5.771 0 0 0];
p2=[0 6.0813 0 0];
p3=[0 0 6.0813 0];
p4=[0 0 0 6.0907];
ee=[];tt=[];xx=[];f11=[];f22=[];f33=[];f44=[];
%
for j1=1:j2
%
if r1(j1,1)>=0 & r1(j1,1)<1;a=0;c=0;d=0;e=0;end;
if r1(j1,1)>=1 & r1(j1,1)<2;a=-.015;c=0;d=0;e=0;end;
if r1(j1,1)>=2 & r1(j1,1)<3;a=0;c=-.02;d=0;e=0;end;
if r1(j1,1)>=3 & r1(j1,1)<4;a=0;c=0;d=-.02;e=0;end;
if r1(j1,1)>=4 & r1(j1,1)<5;a=0;c=0;d=0;e=-.02;end;
tii=ti(j1,1);tff=tf(j1,1);
[tt1,x1]=odd45('fe1',tii,tff,xo,1.0e-6,0,a,c,d,e);
[m1,n1]=size(x1);xo=x1(m1:m1,:);xo=xo';
x1=x1(:,1:8);
e1=5*[[p1;p2;p3;p4] eye(4)]*x1';
ee=[ee;e1'];tt=[tt;tt1];xx=[xx;x1];
f1=6;f2=4;f3=2;f4=0;
```

```
if a==-.015;f1=7;end
if c==-.02; f2=5;end
if d==-.02; f3=3;end
if e==-.02; f4=1;end
f11=[f11;f1*ones(m1,1)];f22=[f22;f2*ones(m1,1)];
f33=[f33;f3*ones(m1,1)];f44=[f44;f4*ones(m1,1)];
end
t3=tt;
x1=xx(:,1);x2=xx(:,2);x3=xx(:,3);x4=xx(:,4);
subplot(221)
axis([0 xm 0 5])
plot(t3,x1,'-',t3,x2,'-.',t3,x3,'--',t3,x4,':')
hold
plot(t3,x1,'-'),xlabel('TIME')
title('SYSTEM STATES'),hold
axis([0 xm -1 8])
plot(t3,f11,'-',t3,f22,'-.',t3,f33,'--',t3,f44,':')
hold
plot(t3,f11,'-'),xlabel('TIME')
title('FAILURE FUNCTIONS'),hold
e=ee;
e1=e(:,1);e2=e(:,2);e3=e(:,3);e4=e(:,4);axis([0 xm -1 1])
plot(t3,e1,'-',t3,e2,'-.',t3,e3,'--',t3,e4,':')
hold
plot(t3,e1,'-'),xlabel('TIME'),title('ROB ERRORS')
hold
%
m1=length(e);
%  ********* weights *************
a1=[-2.2696  8.5391  -3.2950 -2.5377];
a1=[a1  -17.6298   -2.9028   10.5115];
a2=[-11.5401 -9.8273  -0.5848 0.7173];
a2=[a2    8.3441    2.0995    9.0633];
a3=[15.6317 -16.1965   0.4714  -1.4072];
a3=[ a3   2.8586    0.4757    1.7090];
a4=[-1.5534 18.2787   0.5973   -0.0091];
a4=[a4   8.8366   -2.2586  -22.4270];
```

```
w1=[a1;a2;a3;a4];
a1=[-14.6143   -5.8943];
a2=[ 6.4532    8.4886];
a3=[1.5644   11.0521];
a4=[ 8.3970  -14.7202];
wr1434=[w1 [a1;a2;a3;a4]];
%
a1=[ -1.5867   11.7403  -16.6038    1.4582];
a2=[ 14.9443    2.8826    7.6205  -21.5131];
a3=[ -3.7888    0.8799    0.3894    1.8954];
a4=[ -2.9515   -0.7902    1.4499    1.7777];
a5=[-18.6245   -4.0678    2.5003   -3.6738];
a6=[ -3.5271   -1.1462    1.2415    4.7378];
a7=[ 12.9406  -13.6404   -5.3854   14.2183];
a8=[-15.3448   -2.6924    2.9198   -4.3383];
a9=[ -6.2471   -6.3498   -7.6759   14.3213];
wr2434=[a1;a2;a3;a4;a5;a6;a7;a8;a9];
%
%          **** NEURAL NETWORK SECTION ****
n1=4;n2=9;n3=4;
ook=[];
%
for i=1:m1
p11=e(i,:);
[oi,oj,ok]=fpropg(n1,n2,n3,wr1434,wr2434,p11);
ook=[ook;ok'];
end
ok1=ook(:,1);ok2=ook(:,2);ok3=ook(:,3);ok4=ook(:,4);
lll=length(ook);
okr1=ok1+6*ones(lll,1);
okr2=ok2+4*ones(lll,1);
okr3=ok3+2*ones(lll,1);
okr4=ok4+0*ones(lll,1);
axis([0 xm -1 8])
plot(t3,okr1,'-',t3,okr2,'-.',t3,okr3,'--',t3,okr4,':'),hold
plot(t3,okr1,'-'),xlabel('TIME'),
title('ROB/NEUR.NETSIGNALS')
```

```
pause
hold
% ****************************************************
%                  *** EMSF1 ***
%
%  This programe is used to simulate DF/MM1 fault
%  detection scheme (Fast simulation).
%
clc
j2=input('Enter # of failure modes = ');
tw=input('{ti1;ti2;...;tij;tf} = ');
ti=tw(1:j2,:);
tf=tw(2:j2+1,:);
xm=tw(j2+1,:);
clg
r1=5*rand(j2,1)
xo=[2.5;1.5;1;.5;2.5;1.5;1.;.5;0];
% Input data
ee=[];tt=[];xx=[];f11=[];f22=[];f33=[];f44=[];
%
for j1=1:j2
%
if r1(j1,1)>=0 & r1(j1,1)<1;a=0;c=0;d=0;e=0;end;
if r1(j1,1)>=1 & r1(j1,1)<2;a=-.015;c=0;d=0;e=0;end;
if r1(j1,1)>=2 & r1(j1,1)<3;a=0;c=-.02;d=0;e=0;end;
if r1(j1,1)>=3 & r1(j1,1)<4;a=0;c=0;d=-.02;e=0;end;
if r1(j1,1)>=4 & r1(j1,1)<5;a=0;c=0;d=0;e=-.02;end;
tii=ti(j1,1);tff=tf(j1,1);
[t1,x1]=odd45('fe2',tii,tff,xo,1.0e-6,0,a,c,d,e);
[m1,n1]=size(x1);xo=x1(m1:m1,:);xo=xo';
x1=x1(:,1:8);
x=x1(:,1:4);z=x1(:,5:8);e1=(x-z);
ee=[ee;e1];tt=[tt;t1];xx=[xx;x1];
f1=0;f2=0;f3=0;f4=0;
if a==-.015;f1=1;end
if c==-.02;f2=1;end
if d==-.02;f3=1;end
```

```
if e==-.02;f4=1;end
ff1=f1+6;ff2=f2+4;ff3=f3+2;
f11=[f11;ff1*ones(m1,1)];
f22=[f22;ff2*ones(m1,1)];
f33=[f33;ff3*ones(m1,1)];
f44=[f44;f4*ones(m1,1)];
end
t=tt;e=ee;
subplot(221)
x1=xx(:,1);x2=xx(:,2);x3=xx(:,3);x4=xx(:,4);
axis([0 xm 0 5])
plot(t,x1,'-',t,x2,'-.',t,x3,'--',t,x4,':')
hold
plot(t,x1,'-'),xlabel('TIME')
title('SYSTEM STATES')
hold
%
axis([0 xm -1 8])
plot(t,f11,'-',t,f22,'-.',t,f33,'--',t,f44,':')
hold
plot(t,f11,'-'),xlabel('TIME')
title('FAILURE FUNCTIONS')
hold
e1=-50*e(:,1);e2=50*e(:,2);e3=50*e(:,3);e4=50*e(:,4);
e=[e1 e2 e3 e4];
axis([0 xm -1 1])
plot(t,e1,'-',t,e2,'-.',t,e3,'--',t,e4,':')
hold
plot(t,e1,'-'),xlabel('TIME')
title('OBSERVER ERRORS')
hold
%  ****** Neural Net Section ********
a1=[ -4.4782  7.9309   -5.1356   -4.2795];
a1=[a1   -17.3012   -4.7424  11.4082];
a2=[-13.3741 -8.4407   -0.7288    1.7696];
a2=[a2    3.9822    3.2783  10.4737];
a3=[18.9193 -13.2777    1.6448   -1.5672];
```

```
a3=[a3    2.4591    0.1047  -0.7440];
a4=[-1.9945 15.7061    2.3592    1.9428];
a4=[a4   11.7938   -0.8987  -21.9815];
w1=[a1;a2;a3;a4];
%
a1=[-15.7812   -7.2361];
a2=[ 4.4568    8.0792];
a3=[-0.4607    8.4018];
a4=[12.5413  -10.9799];
ws1434=[w1 [a1;a2;a3;a4]];
%
a1=[  -8.7936   18.4845  -16.1528    3.5662];
a2=[  11.7901   -1.1185    6.0211  -20.1709];
a3=[  -8.9484    1.3651   -0.2494    1.9548];
a4=[  -7.7146   -1.4389    1.7844    2.2050];
a5=[ -18.9041   -4.9148    0.4304   -4.2931];
a6=[  -8.0795   -1.3201    1.2409    5.5208];
a7=[  20.0570   -8.2618   -8.3241   11.2661];
a8=[ -18.1122   -5.1992    3.6472   -3.9137];
a9=[  -9.2886   -2.5192   -7.6915   14.5012];
ws2434=[a1;a2;a3;a4;a5;a6;a7;a8;a9];m=length(ee);ook=[];
n1=4;n2=9;n3=4;
for i=1:m
p11=e(i,:);
[oi,oj,ok]=fpropg(n1,n2,n3,ws1434,ws2434,p11);
ook=[ook;ok'];
end
ll=ones(m,1);
ok1=ook(:,1)+6*ll;ok2=ook(:,2)+4*ll;
ok3=ook(:,3)+2*ll;ok4=ook(:,4);
axis([0 xm -1 8])
plot(t,ok1,'-',t,ok2,'-.',t,ok3,'--',t,ok4,':')
hold
plot(t,ok1,'-'),xlabel('TIME')
title('NEURAL NET SIGNALS')
pause
hold
```

```
% ******************************************************
%                  *** EWR03 ***
%
%  This programe is used to simulate RBO/WW2 fault
%  detection scheme (Fast simulation).
%
clc
j2=input('Enter # of Failure modes = ');
tw=input('{ti1;ti2;...;tij;tf} = ');
ti=tw(1:j2,:);
tf=tw(2:j2+1,:);
r1=5*rand(j2,1)
xm=tw(j2+1,:);
xo=[2.5;1.5;1;.5;14.4275;-9.12195;-6.0813;-3.045;0];
clg
% Input data
p1=[-5.771 0 0 0];
p2=[0 6.0813 0 0];
p3=[0 0 6.0813 0];
p4=[0 0 0 6.0907];
ee=[];tt=[];xx=[];f11=[];f22=[];f33=[];f44=[];
%
for j1=1:j2
%
if r1(j1,1)>=0 & r1(j1,1)<1;a=0;c=0;d=0;e=0;end;
if r1(j1,1)>=1 & r1(j1,1)<2;a=-.015;c=0;d=0;e=0;end;
if r1(j1,1)>=2 & r1(j1,1)<3;a=0;c=-.02;d=0;e=0;end;
if r1(j1,1)>=3 & r1(j1,1)<4;a=0;c=0;d=-.02;e=0;end;
if r1(j1,1)>=4 & r1(j1,1)<5;a=0;c=0;d=0;e=-.02;end;
tii=ti(j1,1);tff=tf(j1,1);
[tt1,x1]=odd45('fe1',tii,tff,xo,1.0e-6,0,a,c,d,e);
[m1,n1]=size(x1);xo=x1(m1:m1,:);xo=xo';
x1=x1(:,1:8);
e1=5*[[p1;p2;p3;p4] eye(4)]*x1';
ee=[ee;e1'];tt=[tt;tt1];xx=[xx;x1];
f1=6;f2=4;f3=2;f4=0;
if a==-.015;f1=7;end
```

```
if c==-.02; f2=5;end
if d==-.02; f3=3;end
if e==-.02; f4=1;end
f11=[f11;f1*ones(m1,1)];f22=[f22;f2*ones(m1,1)];
f33=[f33;f3*ones(m1,1)];f44=[f44;f4*ones(m1,1)];
end
t3=tt;
x1=xx(:,1);x2=xx(:,2);x3=xx(:,3);x4=xx(:,4);
subplot(221)
axis([0 xm 0 5])
plot(t3,x1,'-',t3,x2,'-.',t3,x3,'--',t3,x4,':')
hold
plot(t3,x1,'-'),xlabel('TIME')
title('SYSTEM STATES')
hold
axis([0 xm -1 8])
plot(t3,f11,'-',t3,f22,'-.',t3,f33,'--',t3,f44,':')
hold
plot(t3,f11,'-'),xlabel('TIME')
title('FAILURE FUNCTIONS')
hold
e=ee;
e1=e(:,1);e2=e(:,2);e3=e(:,3);e4=e(:,4);
axis([0 xm -1 1])
plot(t3,e1,'-',t3,e2,'-.',t3,e3,'--',t3,e4,':'),hold
plot(t3,e1,'-'),xlabel('TIME'),title('ROB ERRORS')
hold
%
m1=length(e);
%  ********* weights *************
a1=[-2.3155    8.6840 -3.3542  -2.5766];
a1=[a1  -17.9190   -2.9805   10.7563];
a2=[ 0.1322   -0.7992 -0.0051  -0.0012];
a2=[a2   -0.5787    0.0567    0.8237];
a3=[-11.7272   -9.9590 -0.6006   0.7291];
a3=[a3    8.5702    2.0911    9.0746];
a4=[ -0.1170    1.0096 -0.0681  -0.0429];
```

```
a4=[a4    0.3683   -0.1795   -0.8993];
a5=[ 15.8357  -16.3681  0.4653  -1.4190];
a5=[a5    2.9920    0.4370    1.5925];
a6=[ -0.0670    1.4121 -0.0844  -0.0728];
a6=[a6    0.3935   -0.2598   -1.1448];
a7=[-1.6109   18.6504  0.5637  -0.0299];
a7=[a7    8.9876   -2.3522  -22.8333];
a8=[0.0857   -1.9697  0.0833   0.0816];
a8=[a8  -0.3845    0.2537    1.2642];
w1=[a1;a2;a3;a4;a5;a6;a7;a8];
%
a1=[ -14.8547   -5.9540];
a2=[  -0.5365    0.5496];
a3=[   6.6378    8.6047];
a4=[   0.3850   -0.6820];
a5=[   1.6724   11.1712];
a6=[   0.4361   -0.9531];
a7=[   8.5628  -14.9859];
a8=[  -0.4360    1.3027];
wrm1844=[w1 [a1;a2;a3;a4;a5;a6;a7;a8]];
%
%          ****  NEURAL NETWORK SECTION  ****
n1=8;n2=9;n3=4;
ook=[];
%
ook=[0 0 0 0];
for i=2:m1
p11=[e(i,1) e(i-1,1) e(i,2) e(i-1,2)];
p11=[p11 e(i,3) e(i-1,3) e(i,4) e(i-1,4)];
[oi,oj,ok]=fpropg(n1,n2,n3,wrm1844,wrm2844,p11);
ook=[ook;ok'];
end
ok1=ook(:,1);ok2=ook(:,2);ok3=ook(:,3);ok4=ook(:,4);
lll=length(ook);
okr1=ok1+6*ones(lll,1);
okr2=ok2+4*ones(lll,1);
okr3=ok3+2*ones(lll,1);
```

```
okr4=ok4+0*ones(lll,1);
axis([0 xm -1 8])
plot(t3,okr1,'-',t3,okr2,'-.',t3,okr3,'--',t3,okr4,':')
hold
plot(t3,okr1,'-'),xlabel('TIME'),
title('ROB/NEUR.NET SIGNALS')
pause
hold
% ****************************************************
%                *** ENSF3 ***
%
%  This programe is used to simulate DF/NN2 fault
%  detection scheme (Fast simulation).
%
clc
j2=input('Enter # of failure modes = ');
tw=input('{ti1;ti2;...;tij;tf} = ');
ti=tw(1:j2,:);
tf=tw(2:j2+1,:);
xm=tw(j2+1,:);
clg
r1=5*rand(j2,1)
xo=[2.5;1.5;1;.5;2.5;1.5;1.;.5;0];
% Input data
ee=[];tt=[];xx=[];f11=[];f22=[];f33=[];f44=[];
%
for j1=1:j2
%
if r1(j1,1)>=0 & r1(j1,1)<1;a=0;c=0;d=0;e=0;end;
if r1(j1,1)>=1 & r1(j1,1)<2;a=-.015;c=0;d=0;e=0;end;
if r1(j1,1)>=2 & r1(j1,1)<3;a=0;c=-.02;d=0;e=0;end;
if r1(j1,1)>=3 & r1(j1,1)<4;a=0;c=0;d=-.02;e=0;end;
if r1(j1,1)>=4 & r1(j1,1)<5;a=0;c=0;d=0;e=-.02;end;
tii=ti(j1,1);tff=tf(j1,1);
[t1,x1]=odd45('fe2',tii,tff,xo,1.0e-6,0,a,c,d,e);
[m1,n1]=size(x1);xo=x1(m1:m1,:);xo=xo';
x1=x1(:,1:8);
```

```
x=x1(:,1:4);z=x1(:,5:8);e1=x-z;
ee=[ee;e1];tt=[tt;t1];xx=[xx;x1];
f1=0;f2=0;f3=0;f4=0;
if a==-.015;f1=1;end
if c==-.02;f2=1;end
if d==-.02;f3=1;end
if e==-.02;f4=1;end
ff1=f1+6;ff2=f2+4;ff3=f3+2;
f11=[f11;ff1*ones(m1,1)];
f22=[f22;ff2*ones(m1,1)];
f33=[f33;ff3*ones(m1,1)];
f44=[f44;f4*ones(m1,1)];
end
t=tt;
e=ee;
subplot(221)
x1=xx(:,1);x2=xx(:,2);x3=xx(:,3);x4=xx(:,4);
axis([0 xm 0 5])
plot(t,x1,'-',t,x2,'-.',t,x3,'--',t,x4,':'),hold
plot(t,x1,'-'),xlabel('TIME'),title('SYSTEM STATES')
hold
%
axis([0 xm -1 8])
plot(t,f11,'-',t,f22,'-.',t,f33,'--',t,f44,':'),hold
plot(t,f11,'-'),xlabel('TIME'),title('FAILURE FUNCTIONS')
hold
e1=-50*e(:,1);e2=50*e(:,2);e3=50*e(:,3);e4=50*e(:,4);
e=[e1 e2 e3 e4];
axis([0 xm -1 1])
plot(t,e1,'-',t,e2,'-.',t,e3,'--',t,e4,':'),hold
plot(t,e1,'-'),xlabel('TIME'),title('OBSERVER ERRORS'), hold
%  ****** Neural Net Section ********
a1=[-4.6886  8.2023   -5.3074   -4.4316];
a1=[a1  -17.8562   -4.8899  11.9653];
a2=[0.2610 -0.5552    0.1694    0.1416];
a2=[a2    0.2149    0.2144   -0.1464];
a3=[-13.6193 -8.5338   -0.7458    1.7979];
```

```
a3=[a3    3.9973    3.3390  10.7673];
a4=[-1.1607  0.6906   -0.1644   -0.0156];
a4=[a4    0.2243   -0.0672    0.4445];
a5=[19.3340 -13.5058   1.7604   -1.5430];
a5=[a5    2.6293    0.1695   -0.6603];
a6=[0.1163  1.0504   -0.1158   -0.1886];
a6=[a6   -0.0402   -0.3047   -0.0680];
a7=[-1.9013 16.0770    2.3631    1.9454];
a7=[a7   11.9503   -0.9580  -22.4937];
a8=[ 0.9099 -1.3385    0.2851    0.2216];
a8=[a8    0.1182    0.3654   -0.4080];
w1=[a1;a2;a3;a4;a5;a6;a7;a8];
%
a1=[-16.3071   -7.4900];
a2=[ 0.2084    0.4039];
a3=[ 4.4650    8.1829];
a4=[ 0.2038   -0.2969];
a5=[ -0.4051    8.5975];
a6=[ -0.1257   -0.7662];
a7=[ 12.7300  -11.2713];
a8=[ 0.1655    0.8519];
wsm1844=[w1 [a1;a2;a3;a4;a5;a6;a7;a8]];
%
a1=[ -9.2695   18.8437  -16.4250    3.7166];
a2=[ 12.5497   -1.3173    6.2172  -20.6489];
a3=[ -9.2032    1.3495   -0.2560    1.9863];
a4=[ -7.9166   -1.5054    1.8566    2.2448];
a5=[-19.5031   -5.0755    0.4691   -4.4711];
a6=[ -8.3113   -1.3920    1.2675    5.6010];
a7=[ 20.8851   -8.5558   -8.3829   11.3838];
a8=[-18.6399   -5.3830    3.7735   -4.0946];
a9=[ -9.7663   -2.6289   -7.8262   14.7451];
a9=[ -9.7663   -2.6289   -7.8262   14.7451];
wsm2844=[a1;a2;a3;a4;a5;a6;a7;a8;a9];
m=length(ee);ook=[];
n1=8;n2=9;n3=4;
ook=[0 0 0 0];
```

```
for i=2:m
p11=[e(i,1) e(i-1,1) e(i,2) e(i-1,2) e(i,3) e(i-1,3)];
p11=[p11 e(i,4) e(i-1,4)];
[oi,oj,ok]=fpropg(n1,n2,n3,wsm1844,wsm2844,p11);
ook=[ook;ok'];
end
ll=ones(m,1);
ok1=ook(:,1)+6*ll;ok2=ook(:,2)+4*ll;
ok3=ook(:,3)+2*ll;ok4=ook(:,4);
axis([0 xm -1 8])
plot(t,ok1,'-',t,ok2,'-.',t,ok3,'--',t,ok4,':'),hold
plot(t,ok1,'-'),xlabel('TIME'),title('NEURAL NET SIGNALS')
pause
hold
% ******************************************************
%
%                 *** ERROS ***
%
%  This programe is used to simulate RBO/NN3 fault
%  detection scheme (Fast simulation).
clc
j2=input('Enter # of Failure modes = ');
tw=input('{ti1;ti2;...;tij;tf} = ');
ti=tw(1:j2,:);
tf=tw(2:j2+1,:);
r1=5*rand(j2,1)
xm=tw(j2+1,:);
xo=[2.5;1.5;1;.5;14.4275;-9.12195;-6.0813;-3.045;0];
clg
% Input data
p1=[-5.771 0 0 0];
p2=[0 6.0813 0 0];
p3=[0 0 6.0813 0];
p4=[0 0 0 6.0907];
ee=[];tt=[];xx=[];f11=[];f22=[];f33=[];f44=[];
%
for j1=1:j2
```

```
%
if r1(j1,1)>=0 & r1(j1,1)<1;a=0;c=0;d=0;e=0;end;
if r1(j1,1)>=1 & r1(j1,1)<2;a=-.015;c=0;d=0;e=0;end;
if r1(j1,1)>=2 & r1(j1,1)<3;a=0;c=-.02;d=0;e=0;end;
if r1(j1,1)>=3 & r1(j1,1)<4;a=0;c=0;d=-.02;e=0;end;
if r1(j1,1)>=4 & r1(j1,1)<5;a=0;c=0;d=0;e=-.02;end;
tii=ti(j1,1);tff=tf(j1,1);
[tt1,x1]=odd45('fe1',tii,tff,xo,1.0e-6,0,a,c,d,e);
[m1,n1]=size(x1);xo=x1(m1:m1,:);xo=xo';
x1=x1(:,1:8);
e1=5*[[p1;p2;p3;p4] eye(4)]*x1';ee=[ee;e1'];tt=[tt;tt1];xx=[xx;x1];
f1=6;f2=4;f3=2;f4=0;
if a==-.015;f1=7;end
if c==-.02; f2=5;end
if d==-.02; f3=3;end
if e==-.02; f4=1;end
f11=[f11;f1*ones(m1,1)];f22=[f22;f2*ones(m1,1)];
f33=[f33;f3*ones(m1,1)];f44=[f44;f4*ones(m1,1)];
end
t3=tt;
x1=xx(:,1);x2=xx(:,2);x3=xx(:,3);x4=xx(:,4);
subplot(221)
axis([0 xm 0 5])
plot(t3,x1,'-',t3,x2,'-.',t3,x3,'--',t3,x4,':'),hold
plot(t3,x1,'-'),xlabel('TIME'),title('SYSTEM STATES')
hold
axis([0 xm -1 8])
plot(t3,f11,'-',t3,f22,'-.',t3,f33,'--',t3,f44,':'),hold
plot(t3,f11,'-'),xlabel('TIME')
title('FAILURE FUNCTIONS')
hold
e=ee;
e1=e(:,1);e2=e(:,2);e3=e(:,3);e4=e(:,4);
axis([0 xm -1 1])
plot(t3,e1,'-',t3,e2,'-.',t3,e3,'--',t3,e4,':'),hold
plot(t3,e1,'-'),xlabel('TIME'),title('ROB ERRORS');hold
%
```

```
m1=length(e);
%  ********* weights *************
a1=[-2.3387  8.7169  -3.3758   -2.5946];
a1=[a1  -18.0500   -3.0049   10.8950];
a2=[0.1001 -0.3374  -0.0526   -0.0420];
a2=[a2   -0.7266   -0.0314    0.7427];
a3=[0.1908 -0.9628   0.0476    0.0302];
a3=[a3   0.2393    0.0663    0.0496];
a4=[-11.7737 -9.9351  -0.6200    0.7153];
a4=[a4    8.5571    2.0653    9.1639];
a5=[-0.1171  0.7284  -0.0684   -0.0431];
a5=[a5    0.2918   -0.1570   -0.6374];
a6=[-0.1131  0.7326  -0.0680   -0.0433];
a6=[a6    0.2918   -0.1578   -0.6464];
a7=[15.8763 -16.3696  0.4475   -1.4362];
a7=[a7    2.9434    0.4110    1.6658];
a8=[-0.0254  0.9909  -0.0776   -0.0673];
a8=[a8    0.2850   -0.2141   -0.8055];
a9=[-0.0352  1.0250  -0.0791   -0.0682];
a9=[a9    0.2919   -0.2183   -0.8277];
a10=[-1.6211 18.6068   0.5490   -0.0418];
a10=[a10    8.8970   -2.3636  -22.6966];
a11=[0.0204 -1.2194   0.0207    0.0296];
a11=[a11   -0.4287    0.1186    0.9654];
a12=[0.0411 -1.4129   0.0523    0.0526];
a12=[a12   -0.1239    0.1496    0.7449];
w1=[a1;a2;a3;a4;a5;a6;a7;a8;a9;a10;a11;a12];
%
a1=[-14.9689   -5.9636];
a2=[ -0.6386    0.2625];
a3=[  0.1583    0.5320];
a4=[  6.6246    8.6063];
a5=[  0.3011   -0.4896];
a6=[  0.3017   -0.4945];
a7=[  1.6280   11.1837];
a8=[  0.3148   -0.6682];
a9=[  0.3231   -0.6919];
```

```
a10=[ 8.4815  -14.9489];
a11=[-0.4295    0.8281];
a12=[-0.1777    0.9082];
wrm11254=[w1 [a1;a2;a3;a4;a5;a6;a7;a8;a9;a10;a11;a12]];
%
a1=[  -1.5732   12.0046  -16.8957    1.3138];
a2=[  15.4310    3.0247    7.8834  -22.1665];
a3=[  -3.8126    0.8905    0.3972    1.9020];
a4=[  -2.9626   -0.7983    1.4769    1.7963];
a5=[ -19.0932   -4.2124    2.5520   -3.9888];
a6=[  -3.5528   -1.1719    1.2461    4.7797];
a7=[  13.3321  -13.8726   -5.4899   14.5218];
a8=[ -15.7121   -2.8109    2.9755   -4.6872];
a9=[  -6.4152   -6.4949  ,-7.8545   14.5461];
wrm21254=[a1;a2;a3;a4;a5;a6;a7;a8;a9];
%          ****  NEURAL NETWORK SECTION  ****
n1=12;n2=9;n3=4;
ook=[];
%
ook=[0 0 0 0;0 0 0 0];
for i=3:m1
p11=[e(i,1) e(i-1,1) e(i-2,1) e(i,2) e(i-1,2) e(i-2,2)];
p11=[p11 e(i,3) e(i-1,3) e(i-2,3) e(i,4)];
p11=[p11 e(i-1,4) e(i-2,4)];
[oi,oj,ok]=fpropg(n1,n2,n3,wrm11254,wrm21254,p11);
ook=[ook;ok'];
end
ok1=ook(:,1);ok2=ook(:,2);ok3=ook(:,3);ok4=ook(:,4);
lll=length(ook);
okr1=ok1+6*ones(lll,1);okr2=ok2+4*ones(lll,1);
okr3=ok3+2*ones(lll,1);okr4=ok4+0*ones(lll,1);
axis([0 xm -1 8])
plot(t3,okr1,'-',t3,okr2,'-.',t3,okr3,'--',t3,okr4,':')
hold
plot(t3,okr1,'-'),xlabel('TIME'),
title('ROB/NEUR.NET SIGNALS')
pause
```

```
hold
% ****************************************************
%                  *** EMSF5 ***
%
%  This programe is used to simulate DF/HH3 fault
%  detection scheme (Fast simulation).
%
clc
j2=input('Enter # of failure modes = ');
tw=input('{ti1;ti2;...;tij;tf} = ');
ti=tw(1:j2,:);
tf=tw(2:j2+1,:);
xm=tw(j2+1,:);
clg
r1=5*rand(j2,1)
xo=[2.5;1.5;1;.5;2.5;1.5;1.;.5;0];
% Input data
ee=[];tt=[];xx=[];f11=[];f22=[];f33=[];f44=[];
%
for j1=1:j2
%
if r1(j1,1)>=0 & r1(j1,1)<1;a=0;c=0;d=0;e=0;end;
if r1(j1,1)>=1 & r1(j1,1)<2;a=-.015;c=0;d=0;e=0;end;
if r1(j1,1)>=2 & r1(j1,1)<3;a=0;c=-.02;d=0;e=0;end;
if r1(j1,1)>=3 & r1(j1,1)<4;a=0;c=0;d=-.02;e=0;end;
if r1(j1,1)>=4 & r1(j1,1)<5;a=0;c=0;d=0;e=-.02;end;
tii=ti(j1,1);tff=tf(j1,1);
[t1,x1]=odd45('fe2',tii,tff,xo,1.0e-6,0,a,c,d,e);
[m1,n1]=size(x1);xo=x1(m1:m1,:);xo=xo';
x1=x1(:,1:8);
x=x1(:,1:4);z=x1(:,5:8);e1=x-z;
ee=[ee;e1];tt=[tt;t1];xx=[xx;x1];
f1=0;f2=0;f3=0;f4=0;
if a==-.015;f1=1;end
if c==-.02;f2=1;end
if d==-.02;f3=1;end
if e==-.02;f4=1;end
```

```
ff1=f1+6;ff2=f2+4;ff3=f3+2;
f11=[f11;ff1*ones(m1,1)];
f22=[f22;ff2*ones(m1,1)];
f33=[f33;ff3*ones(m1,1)];
f44=[f44;f4*ones(m1,1)];
end
t=tt;
e=ee;
subplot(221)
x1=xx(:,1);x2=xx(:,2);x3=xx(:,3);x4=xx(:,4);
axis([0 xm 0 5])
plot(t,x1,'-',t,x2,'-.',t,x3,'--',t,x4,':'),hold
plot(t,x1,'-'),xlabel('TIME'),title('SYSTEM STATES')
hold
%
axis([0 xm -1 8])
plot(t,f11,'-',t,f22,'-.',t,f33,'--',t,f44,':'),hold
plot(t,f11,'-'),xlabel('TIME'),title('FAILURE FUNCTIONS')
hold
e1=-50*e(:,1);e2=50*e(:,2);e3=50*e(:,3);e4=50*e(:,4);
e=[e1 e2 e3 e4];
axis([0 xm -1 1])
plot(t,e1,'-',t,e2,'-.',t,e3,'--',t,e4,':'),hold
plot(t,e1,'-'),xlabel('TIME'),title('OBSERVER ERRORS'), hold
%  ****** Neural Net Section ********
a1=[-4.61 8.1 -5.24  -4.37 -17.63 -4.83 11.74 -16.1 -7.39];
a2=[0.1 -0.24 0.06 .05 0.04 0.09 .01 .03 .17];
a3=[0.16 -0.36 0.12 .11 0.23 0.15 -.14 .22 .27];
a4=[-13.53 -8.46 -0.75 1.78 3.97 3.83 10.66 4.44 8.12];
a5=[-.66 0.39 -.1 -.01 0.11 -.04 .28 .09 -.17];
a6=[-.44 0.35 -.07 -.01 0.1 -.04 .19 .09 -.18];
a7=[19.13 -13.38 1.69 -1.56 2.53 .13 -0.65 -.46 8.5];
a8=[.06 .54 -.06 -0.1 -.02 -.15 -.01 -.07 -.39];
a9=[.06 .56 -.07 -0.1 -.02 -.16 -.03 -.07 -.40];
a10=[-1.94 15.93 2.34 1.93 11.83 -.94 -22.22 12.6 -11.16];
a11=[.48 -.64 .12 .09 0 .17 -.18 .02 .4];
a12=[.37 -.67 .14 .12 0.09 .19 -.2 .11 .44];
```

```
wsm11254=[a1;a2;a3;a4;a5;a6;a7;a8;a9;a10;a11;a12];
a1=[-9.04  18.67  -16.3  3.65];
a2=[12.23  -1.2  6.15  -20.43];
a3=[-9.08  1.35  -0.25  1.97];
a4=[-7.82  -1.48  1.83  2.22];
a5=[-19.21  -5  .46  -4.39];
a6=[-8.2  -1.37  1.26  5.56];
a7=[20.5  -8.45  -8.35  11.32];
a8=[-18.38  -5.29  3.73  -4.02];
a9=[-9.55  -2.6  -7.76  14.62];
wsm21254=[a1;a2;a3;a4;a5;a6;a7;a8;a9];
m=length(ee);ook=[];
n1=12;n2=9;n3=4;
ook=[0 0 0 0;0 0 0 0];
for i=3:m
p11=[e(i,1) e(i-1,1) e(i-2,1) e(i,2) e(i-1,2) e(i-2,2)];
p11=[p11 e(i,3)  e(i-1,3) e(i-2,3) e(i,4)];
p11=[p11 e(i-1,4) e(i-2,4)];
[oi,oj,ok]=fpropg(n1,n2,n3,wsm11254,wsm21254,p11);
ook=[ook;ok'];
end
ll=ones(m,1);
ok1=ook(:,1)+6*ll;ok2=ook(:,2)+4*ll;
ok3=ook(:,3)+2*ll;ok4=ook(:,4);
axis([0 xm -1 8])
plot(t,ok1,'-',t,ok2,'-.',t,ok3,'--',t,ok4,':'),hold
plot(t,ok1,'-'),xlabel('TIME')
title('NEURAL NET SIGNALS')
pause
hold
%
% ****************************************************
%
%              *** EX571 ***
%
%  This program is used to train a three layer neural
%  networks.  It uses the back-propagation algorithm of
```

```
%  Rumelhart et al. (1986).
%
%  Inputs
%  -------
%    p : Input patterns.
%    d : output patterns.
%    n1: Number of nodes in the input layer.
%    n2: Number of nodes in the first hidden layer.
%    n3: Number of nodes in the second hidden layer.
%    n4: Number of nodes in the output layer.
%    w1: Is the initial weights between the input and
%        hidden layers.
%    w2: Is the initial weights between the hidden and
%        ouput layers.
% beata: The learning rate constant.
% Alpha: The momentum rate constant.
%
%  Outputs
%  -------
%    w1: Is the final weights between the input and
%        hidden layers.
%    w2: Is the final weights between the hidden and
%        ouput layers.
%    e : error versus number of iterations.
%        .
%
%
%
%
%          ****  NEURAL NETWORK PROGRAM  ****
%
%
%
n1=4;n2=9;n3=4;ii=0;
rrr=input('initialize w1 & w2   rrr<5')
if rrr<5;
w1=2*(rand(n1,n2)-ones(n1,n2));
```

```
w2=2*(rand(n2,n3)-ones(n2,n3));
end
dw1o=zeros(n1,n2);dw2o=zeros(n2,n3);
%alpha=input('Momentum factor = ');
n=input('NUMBER OF ITERATIONS = ');
%
bb=input('input learning rate vector')
alpha=input('input momentum rate ')
count = 0
m=size(p);m=m(1,1);
m4=size(bb);m4=m4(1,1);
e=[];
%       ************** MAIN LOOP  1 **************
for k=1:n;
dd1=zeros(n1,n2);dd2=zeros(n2,n3);
e2=[];
%       ************** MAIN LOOP  2 **************
for i=1:m4;count=count+1
beta=bb(i);
%     ************** SECONDARY LOOP **************
for j=1:m;
d11=d(j,:);d11=d11';
[oi,oj,ok]=fpropg(n1,n2,n3,w1,w2,p(j,:));
%
%       ***** Back propgation segment *****
%
[dj,dk]=prop(n1,n2,n3,w1,w2,oj,ok,d11);
%
dw2=[];dw1=[];
for l=1:n2;
dw21=beta*dk'*oj(l);dw2=[dw2;dw21];end
%
for r=1:n1;
dw11=beta*dj'*oi(r);dw1=[dw1;dw11];end
%
dd1=dd1+dw1;
dd2=dd2+dw2;
```

```
ee2=d11-ok;e2=[e2;ee2];
end
w1=w1+dd1+alpha*dw1o;w2=w2+dd2+alpha*dw2o;
dw1o=dd1;dw2o=dd2;dd1=zeros(n1,n2);dd2=zeros(n2,n3);
ie=e2'*e2;e2=[];e=[e;ie];error=ie
iii=[iii;ii];
%plot(iii,100*e),xlabel('Iteration number')
%ylabel('Objective Function')
end
end
%
ook=[];
for k=1:1:m;
[oi,oj,ok]=fpropg(n1,n2,n3,w1,w2,p(k,:));
ook=[ook;ok'];end
%
% ******************************************************
%    SUBROUTINES USED BY THE PREVIOUS PROGRAMS
% ******************************************************
%
%                *** ODD45 ***
%
%  This subroutine is used to bu all programs used in
%  this thesis.  It is basically utilized to integrate
%  the differential equation used in the above programs.
%
%
function [tout,yout]=odd45(F,t0,tfinal,y0,tol,trace,a,c,d,e)
%ODE45  Integrate a system of ordinary differential
%equations using        4th and 5th order formulas.
%
% [tout, yout] = ode45(F, t0, tfinal, y0, tol, trace)
%
% INPUT:
% F     - String containing name of user-supplied problem
%         description.
%         Call: yprime = fun(t,y) where F = 'fun'.
```

```
%         t      - Time (scalar).
%         y      - Solution column-vector.
%         yprime - Returned derivative column-vector;
%         yprime(i) = dy(i)/
% t0    - Initial value of t.
% tfinal- Final value of t.
% y0    - Initial value column-vector.
% tol   - The desired accuracy. (Default: tol = 1.e-6).
% trace - If nonzero, each step is printed.
%         (Default: trace = 0).
%
% OUTPUT:
% tout  - Returned integration time points (row-vector).
% yout  - Returned solution, one solution column-vector
%         per tout-value.
%
% The result can be displayed by: plot(tout, yout).

%   C.B. Moler, 3-25-87.
%   Copyright (c) 1987 by the MathWorks, Inc.
%   All rights reserved.

% The Fehlberg coefficients:alpha = [1/4  3/8  12/13  1  1/2]';
beta  = [ [    1      0      0     0      0    0]/4
          [    3      9      0     0      0    0]/32
          [ 1932  -7200   7296     0      0    0]/2197
          [ 8341 -32832  29440  -845      0    0]/4104
          [-6080  41040 -28352  9295  -5643  0]/20520 ]';
gamma=[[902880 0 3953664 3855735 -1371249 277020]/7618050
 [ -2090  0    22528    21970    -15048  -27360]/752400 ]';
pow = 1/5;
if nargin < 6, trace = 0; end
if nargin < 5, tol = 1.e-6; end
% Initialization
t = t0;
hmax = (tfinal - t)/5;
hmin = (tfinal - t)/20000;
```

```
h = (tfinal - t)/100;
y = y0(:);
f = y*zeros(1,6);
tout = t;
yout = y.';
tau = tol * max(norm(y, 'inf'), 1);

if trace
   clc, t, h, y
end

% The main loop
   while (t < tfinal) & (h >= hmin)
      if t + h > tfinal, h = tfinal - t; end
      % Compute the slopes
      f(:,1) = feval(F,t,y,a,c,d,e);
      for j = 1:5
f(:,j+1)=feval(F, t+alpha(j)*h, y+h*f*beta(:,j),a,c,d,e);
      end
      % Estimate the error and the acceptable error
      delta = norm(h*f*gamma(:,2),'inf');
      tau = tol*max(norm(y,'inf'),1.0);
 % Update the solution only if the error is acceptable
      if delta <= tau
         t = t + h;
         y = y + h*f*gamma(:,1);
         tout = [tout; t];
         yout = [yout; y.'];
      end
      if trace
         home, t, h, y
      end
      % Update the step size
      if delta ~= 0.0
         h = min(hmax, 0.8*h*(tau/delta)^pow);
      end
   end;
```

```
   if (t < tfinal)
      disp('SINGULARITY LIKELY.')
      t
   end
% ****************************************************
%
%                  *** PROP ***
%
% This program is used as a subroutine to compute delta
% weights change for a three layer neural network
%
%  Inputs
%  --------
%    d : Target values.
%    n1: Number of nodes in the input layer.
%    n2: Number of nodes in the hidden layer.
%    n3: Number of nodes in the output layer.
%    w1: Is the weight between input layer and first
%        hidden layer.
%    w2: Is the weight between first hidden layer and
%        output layer.
%    oj: Computed outputs at the hidden layer.
%    ok: Computed outputs at the output layer.
%        .
%  Outputs
%  -------
%    dj: Is the delta weights change in the input/hidden
%        layer.
%    dk: Is the delta weights change in the hidden/output
%        layer.
%
function [dj,dk]=prop(n1,n2,n3,w1,w2,oj,ok,d)
% This program simulates the back propgation
dk=[];
dk=diag(d'-ok',0)*diag(ok',0)*(ones(n3,1)-ok);
dj=[];
```

```
for j=1:n2;
dj1=oj(j)*(1-oj(j))*w2(j:j,1:n3)*dk;
dj=[dj;dj1];end;
end
% ******************************************************
%
%                *** FPROPG ***
%
%  This program is used as a subroutine to compute the
%  output for a three layer neural network.
%
%  Inputs
%  -------
%   u : Input patterns.
%   n1: Number of nodes in the input layer.
%   n2: Number of nodes in the hidden layer.
%   n3: Number of nodes in the output layer.
%   w1: Is the weight between input layer and hidden layer.
%   w2: Is the weight between hidden layer and output layer.
%
%  Outputs
%  -------
%   a: Computed outputs at the input layer.
%   b: Computed outputs at the hidden layer.
%   c: Computed outputs at the output layer.
%
function [a,b,c]=propg(n1,n2,n3,w1,w2,u);
% This programe calulate the outputs of neural network arch.
a=[];b=[];c=[];u=u';
for i=1:n1;oi1=1.0/(1.0+exp(-u(i)));a=[a;oi1];end;
for j=1:n2;oj1=1.0/(1.0+exp(-a'*w1(1:n1,j)));b=[b;oj1];end;
for k=1:n3;ok1=1.0/(1.0+exp(-b'*w2(1:n2,k)));c=[c;ok1];end;
end
% ******************************************************
%
%                *** Fe1 ***
%
```

```
%  This program is a subrotine used to store the model
%  and the robust observer equations.
%
%
function [xd]=fe1(t,x,a,c,d,e)
b=.2;u=1;f=-2;
a1=-b*sqrt(x(1)-x(2))+.2*u+a*(x(1))^.5;
a2=b*sqrt(x(1)-x(2))-b*sqrt(x(2)-x(3))+c*(x(2))^.5;
a3=b*sqrt(x(2)-x(3))-b*sqrt(x(3)-x(4))+d*(x(3))^.5;
a4=b*sqrt(x(3)-x(4))-(b-e)*sqrt(x(4));
a5=f*x(5)+10.9649*x(1)+.5771*x(2)+.5771*u;
a6=f*x(6)-.6081*x(1)-10.9464*x(2)-.6081*x(3);
a7=f*x(7)-.6081*x(2)-10.9464*x(3)-.6081*x(4);
a8=f*x(8)-.6091*x(3)-10.9632*x(4);
a9=-5*x(9)+5*x(1);
xd=[a1;a2;a3;a4;a5;a6;a7;a8;a9];
%
% ****************************************
%
%                *** Fe2 ***
%
%  This program is a subrotine used to store the model
%  and the detection filter equations.
%
function xd=fe2(t,x,a,c,d,e)
b=.2;u=1;
a1=-b*sqrt(x(1)-x(2))+.2*u+a*(x(1))^.5;
a2=b*sqrt(x(1)-x(2))-b*sqrt(x(2)-x(3))+c*(x(2))^.5;
a3=b*sqrt(x(2)-x(3))-b*sqrt(x(3)-x(4))+d*(x(3))^.5;
a4=b*sqrt(x(3)-x(4))-b*sqrt(x(4))+e*(x(4))^.5;
a5=-2*x(5)+.1*u+1.9*x(1)+.1*x(2);
a6=-2*x(6)+.1*x(1)+1.8*x(2)+.1*x(3);
a7=-2*x(7)+.1*x(2)+1.8*x(3)+.1*x(4);
a8=-2*x(8)+.1*x(3)+1.8*x(4);
a9=-5*x(9)+5*x(1);
xd=[a1;a2;a3;a4;a5;a6;a7;a8;a9];
```

Bibliography

[1] Himmelblau, D.M., *Fault Detection and Diagnosis in Chemical and Petrochemical Processes*, Elsevier, New York, 1978.

[2] Hoskins, J.C., and D.M. Himmelblau, *"Artificial Neural Networks Models of Knowledge Representation in Chemical Engineering"*, Comp. Chem. Eng., Vol. 12, PP. 881-890, 1988.

[3] Mehra, R.K., and I. Peshon, *"An Innovation Approach to Fault Detection and Diagnosis in Dynamic Systems"*, Automatica, Vol. 7, PP. 637-640, 1971.

[4] Frank, P.M., *"Fault Diagnosis in Dynamic Systems Using Analytical and Knowledge-based Redundancy-A Survey and Some New Results"*, Automatica, Vol. 26, PP. 459-474, 1990.

[5] Basseville, M., and A. Benveniste, *Detection of Abrupt Changes in Signals and Dynamical Systems*, Lectures Note in Control and Information Science, Vol. 77, Springer-Verlag, Berlin, 1985.

[6] Willsky, A.S., *"A Survey of Design Methods fo Failure Detection in Dynamic Systems"*, Automatica, Vol. 12, PP. 601-611, 1976.

[7] Isermann, R., *"Process Fault Detection Based on Modeling and Estimation Methods-A Survey"*, Automatica, Vol. 20, PP. 387-404, 1984.

[8] Ge, W., and C.Z. Fang, *"Detection of Fault Components via Robust Observation"*, Int. J. Control, Vol. 47, PP. 581-599, 1988.

[9] Beard, R.V., *"Failure Accommodation in Linear Systems Through Self-Reorganization"*, Ph.D. Thesis, Department of Aeronautics and Astronautics, MIT, Cambridge, MA., Feb. 1971.

[10] Jones, H.L., *"Failure Detection in Linear Systems"*, Ph.D. Thesis, Department of Aeronautics and Astronautics, MIT, Cambridge, MA., Sept. 1971.

[11] Viswanadham, N.V., V.S. Sarma, and M.G. Singh, *Reliability of Computer and Control Systems*, North Holland Systems and Control Series, Vol. 8, North Holland, 1987.

[12] Frank, P.M., and L. Keller, *"Sensitivity Discriminating Observer Design for Instrument Failure Detection"*, IEEE Trans. on Aerospace Electron. Syst., Vol. AES-16, PP. 460-467, 1980.

[13] Venkatasubramanian, V., and K. Chen, *"A Neural Network Methodology for Process Fault Detection"*, AIChE J., Vol. 35, PP. 1993-2002, 1989.

[14] Rumelhart, D.E., J. McClelland, and the PDP Research Group, *Parallel Distributed Processing: Exploration in the Microstructure of Cognition-Foundation*, MIT Press, Cambridge, MA., 1986.

[15] Lippmann, R.P., *"An Introduction to Computing with Neural Nets"*, IEEE ASSP Mag, Vol. 4, PP. 4-22, 1987.

[16] Pao, Y.H., *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley, Reading, 1989.

[17] Strom, T., *"On Logarithmic Norms"*, SIAM J. Num. Anal., Vol. 12, PP. 741-753, 1975.

[18] Ezzine, J., *"Parameter Perturbations in Digital Control Systems"*, M.S. Thesis, Department of Electrical and Computer Engineering, University of Alabama, Huntsville, May 1985.

[19] Hoskins, J.C., K.M. Kaliyur, and D.M. Himmelblau, *"Fault Diagnosis in Complex Plants Using Artificial Neural Networks"*, AIChE J., Vol. 37, PP. 137-141, 1991.

[20] Watanaba, K., I. Matsuura, M. Abe, M. Kubota, and D.M. Himmelblau, *"Incipient Fault Diagnosis of Chemical Processes via Artificial Neural Networks"*, AIChE J., Vol. 35, PP. 1803-1812, 1989.

[21] Van Loan, C., *"The Sensitivity of The Matrix Exponential"*, SIAM J. Num. Anal., Vol. 14, PP. 971-981, 1977.

[22] Caudill, M. and C. Butler, *Naturally Intelligent Systems*, The MIT Press, Cambridge, Massachusetts, 1989.

[23] Hoskins, J.C. et al., *" Fault Diagnosis in Complex Plants Using Artificial Neural Networks"*, AIChE J., Vol. 37, PP. 137-141, 1991.

[24] Ungar, L.H., B.A. Powell and S.N. Kamens, *" adaptive networks for Fault Diagnosis and Process Control"*, Comp. Chem. Eng., Vol. 14, PP. 561-572, 1990.

[25] Venkatasubramanian, V. et al., *"Process Fault Detection and Diagnosis Using Neural Networks-I. Steady-State Processes"*, Comp. Chem. Eng., Vol. 14, PP. 699-712, 1990.

[26] Kramer, M.A. and J.A. Leonard, *"Diagnosis Using Backpropagation Neural Networks-Analysis and Criticism"*, Comp. chem. Eng., Vol. 14, PP. 1323-1338, 1990.

[27] Cooper, D.J. and A.M. Lalonde, *"Process Behavior Diagnostics and Adaptive Process Control"*, Comp. Chem. Eng., Vol. 14, PP. 541-549, 1990.

[28] Duyar, A. and W. Merrill, *"A Failure Diagnosis System Based on a Neural Network Classifier for the Space Shuttle Main Engine"*, Proc.of the 29th Conf. on Dec. and Contr., PP. 2391-2400, 1990.

[29] Yao, S.C. and E. Zafiriou, *"Control System Sensor Failure Detection via Networks of Localized Receptive Fields"*, Proc. Amer. Contr. Conf., PP. 2472-2477, 1990.

[30] Sinnasamy, R. et al., *" Use of Neural Networks for Sensor Failure Detection in Control Systems"*, IEEE Contr. Syst. Mag., Vol. 10, PP. 49-55, April 1990.

Vita

Mufeed Ahmed Saleh AL-Ghumgham

Born at Dhahran, Saudi Arabia

Received Bachelor's degree in Chemical Engineering from King Fahd University of Petroleum & Minerals (KFUPM), Saudi Arabia in Sept. 1988.

Completed Master's degree requirements at KFUPM, Dhahran, Saudi Arabia in July 1992.